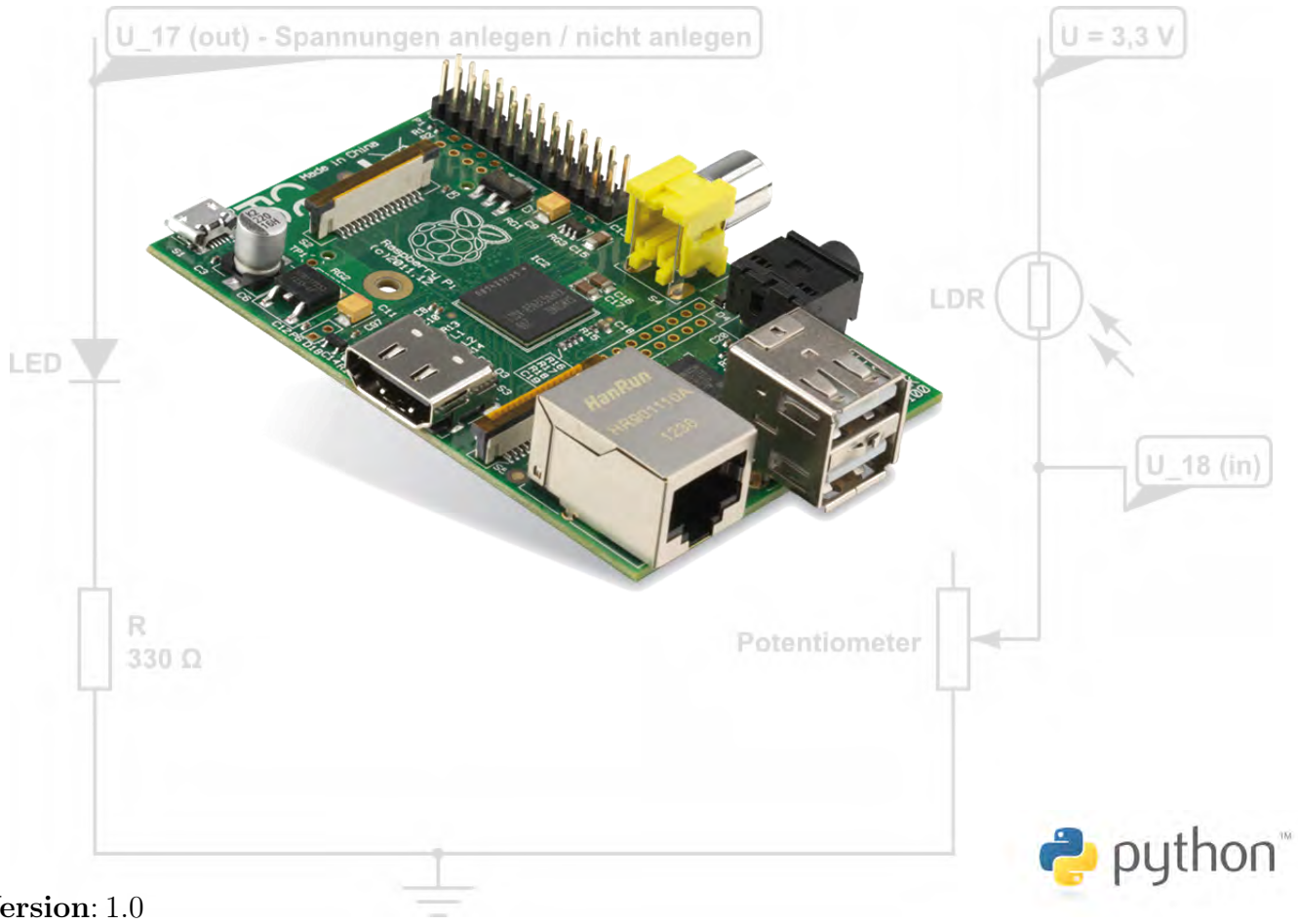
TfK Technik für Kinder e.V.
Begeistern durch Machen

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Modul:

Rechnernetze und Physical Computing

Werkzeug: Raspberry Pi



Version: 1.0

Dieses Modul wurde unter Leitung von Ute Heuer an der Lehr- und Forschungseinheit Mathematik und Informatik (LMI) der Universität Passau in Kooperation mit dem Verein „TfK - Technik für Kinder e.V.“ im Rahmen des Projekts Technikgruppen an Schulen - Kinder für Technik begeistern erarbeitet.
Bei der Entwicklung der Aufgaben des Kapitels *Physical Computing* wurden wir von den studentischen Hilfskräften Stefan Brand und Torben Schnuchel unterstützt.

Mehr zu diesem Projekt erfahren Sie auf der Internetseite

http://www.fim.uni-passau.de/fim/fakultaet/lehrstuehle-professuren-und-fachgebiete-der-fim/didaktik-der-informatik/projekte/technikgruppen.html .

Ute Heuer

Didaktik der Informatik
ute.heuer@uni-passau.de

Wolfgang Pfeffer

Didaktik der Informatik / Mathematik
wolfgang.pfeffer@uni-passau.de

Modulübersicht

Im Rahmen des Kooperationsprojekts zwischen der Universität Passau und dem Verein Technik für Kinder e.V. sind folgende Module entstanden:

Modul Roboter-Programmierung

Dieses Modul bietet einen sehr kurzen Einstieg in die Programmierung eines Lego EV3 Roboters mit der Lego Mindstorms Education Software. Neben Hardware- und Software Anforderungen wird die Entwicklungsumgebung vorgestellt und anschließend Einstiegsaufgaben zu Motoren, Sensoren sowie vermischte Aufgaben bereitgestellt.

Modul App-Entwicklung auf Mobile Devices

Dieses Modul thematisiert umfassend die App-Entwicklung auf Mobile Devices mit dem AppInventor, einer graphischen Programmierumgebung. Ein einfacher und handlungsorientierter Zugang zu dieser Thematik steht dabei im Vordergrund. Neben Hardware- und Softwareanforderungen wird eine ausführliche Installationsanleitung angeboten. Die Einführungsaufgabe 'Schritt für Schritt zur ersten eigenen App' ermöglicht eine Einführung in den Umgang mit dem AppInventor. Gleichzeitig werden fast alle wichtigen Elemente behandelt. Anschließend umfasst das Modul 10 Aufgaben mit unterschiedlichem Schwierigkeitsgrad sowieso ausführlichen Lösungen. Exkurse zu Themen wie GPS und Dijkstra-Algorithmus runden die Handreichung ab.

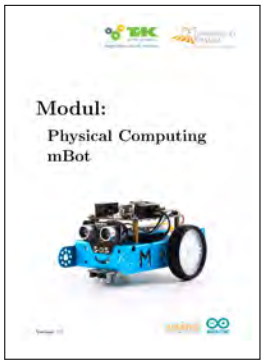
Modul App-Entwicklung mit Java Android

Dieses Modul baut thematisch auf dem Modul 'App-Entwicklung auf Mobile Devices' auf, anstelle der graphischen Programmierung werden die Applikationen nun mit Java Android entwickelt. Da die Einstiegshürde in Java Android vergleichsweise hoch ist, sind grundlegende Java-Kenntnisse unumgänglich. Es wird ein sehr ausführlicher sowie kleinschrittiger Einstieg in die Programmierung mit Java Android geboten. Anschließend stehen verschiedene Projekte mit umfangreichen Lösungen zur Auswahl (z.B. Vokabel-App, Quiz-App, 2048-App, Datenspeicher oder LegoPilot-App). Dabei werden wichtige Konzepte der Sekundarstufe II behandelt.

Modul Physical Computing

Dieses Modul kombiniert Elemente aus der Physik und der Informatik und basiert auf dem Einplatinencomputer Raspberry Pi. Zunächst geht es darum, (einfache) Schaltungen zu erstellen und mit konstanter Spannungsquelle zu arbeiten. Hierbei wird mit verschiedenen Schaltungselementen experimentiert (z.B. LED, Widerstand, LDR, Poti, Servomotor). Anschließend soll an manchen Pins gezielt Spannung angelegt oder nicht angelegt werden. Dazu werden die GPIO-Pins mit der Programmiersprache Python konfiguriert und angesteuert. Die Handreichung beinhaltet ein eigenes Kapitel zu Python, in welchem man sich anhand kleiner Aufgaben mit der Syntax der Programmiersprache vertraut machen kann.

Modul Physical Computing mBot



Dieses Modul kann als Schnittstellenmodul zwischen Roboter-Programmierung und Physical Computing gesehen werden. Bei mBot handelt es sich um ein handliches Roboterfahrzeug, das mit verschiedenen Sensoren und Motoren ausgestattet ist, die mit dem Mikrocontroller Arduino verbunden werden können. Im Bezug auf die Programmierung bietet das Modul zwei Zugangswege: Ähnlich zur App-Entwicklung mit dem AppInventor kann der mBot mithilfe der graphischen Programmierumgebung Scratch angesteuert werden. Ein zweiter Teil des Moduls bietet die Programmierung des mBot mit Hilfe von Arduino-C.

Quelle des Bildes: https://www.colourbox.de/vektor/buch-vektor-5814145

Inhaltsverzeichnis

Kapitel 1

Über das Modul

Rechnernetze und Physical Computing - der Titel des Moduls weist auf die zwei großen Bereiche hin, mit denen sich diese Handreichung befasst.

In der Version 1.0 umfasst die Handreichung nur den Themenbereich „Physical Computing“.

Was steckt hinter dem Begriff „Physical Computing“? Dieser setzt sich aus den beiden Wörtern Physik und Computing zusammen. Physik deswegen, weil es in den ersten Aufgaben darum geht, (einfache) Schaltungen zu erstellen und mit konstanter Spannungsquelle zu arbeiten. Wir werden Schaltungen nicht löten, sondern Breadboards verwenden. Weiter werden wir mit verschiedenen Schaltungselementen (z.B. LED, Widerstand, LDR, Potentiometer, Servomotor) experimentieren, Spannungsabfall messen und mit physikalischen Schaltplänen arbeiten.

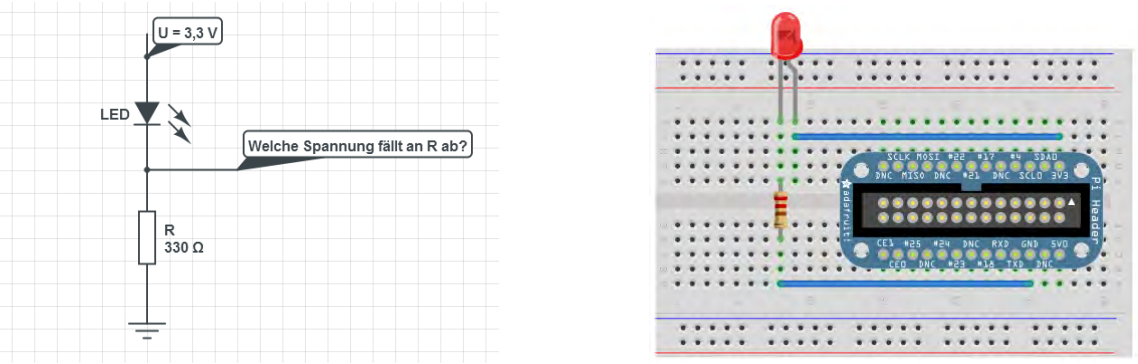


Abbildung 1.1: Beispiele für einen physikalischen Schaltplan (links) und dessen Umsetzung auf einem Breadboard (rechts).

Wir wollen allerdings nicht die ganze Zeit mit einer konstanten Spannungsquelle arbeiten, sondern auch gezielt an manchen Pins Spannung anlegen bzw. eben nicht anlegen oder auch Spannung messen. An diesem Punkt kommt „Computing“ ins Spiel - wir werden mithilfe der Programmiersprache Python einzelne GPIO-Pins des Raspberry Pi konfigurieren und ansteuern.

Abbildung 1.2: GPIO-Pins werden wir mithilfe der Programmiersprache Python ansteuern.

In ersten Tests mit Schülerkursen haben wir die Erfahrung gemacht, dass die Einstiegshürde mit Physik und der vielen Schülern unbekannten Programmiersprache Python relativ groß ist. Um diese Hürde abzuschwächen, haben wir ein eigenes Kapitel „Programmiersprache Python“ entwickelt, in dem man sich anhand kleinerer Aufgaben mit der Syntax von Python vertraut machen kann.

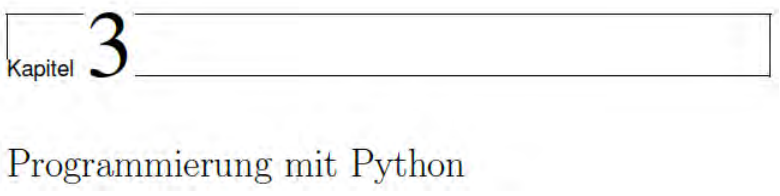


Abbildung 1.3: Die Handreichung beinhaltet ein eigenes Kapitel „Programmierung mit Python“, in dem man sich zunächst mit der vielen unbekannten Sprache Python und deren Syntax vertraut machen kann.

Die Raspberry Pis kann man auch verwenden, um ein eigenes Labornetz zu erstellen. In diesem Netz kann man verschiedenen Aufgaben zur Schichtenarchitektur bearbeiten und somit ein besseres Verständnis von dem schwer fassbaren Begriff „Internet“ zu erhalten. Schülerinnen und Schüler lernen dabei verschiedene Protokolle (`HTTP, ICMP, TCP, etc.`) kennen, konfigurieren eine Firewall oder richten einen Proxy ein. In der aktuellen Modulversion ist dieses Kapitel allerdings noch nicht umgesetzt.

Uns ist Ihre Meinung als Betreuer der Technikgruppen sehr wichtig. Wir würden uns freuen, wenn Sie Anregungen oder etwaige Verbesserungsvorschläge an uns weiterleiten würden.
Vorschläge für neue Aufgaben und Erfahrungen aus dem Einsatz des Moduls in den Technikgruppen sind für uns ebenfalls sehr hilfreich!

Bitte senden Sie Ihre Erfahrungen und Anregungen an `wolfgang.pfeffer@uni-passau.de`.

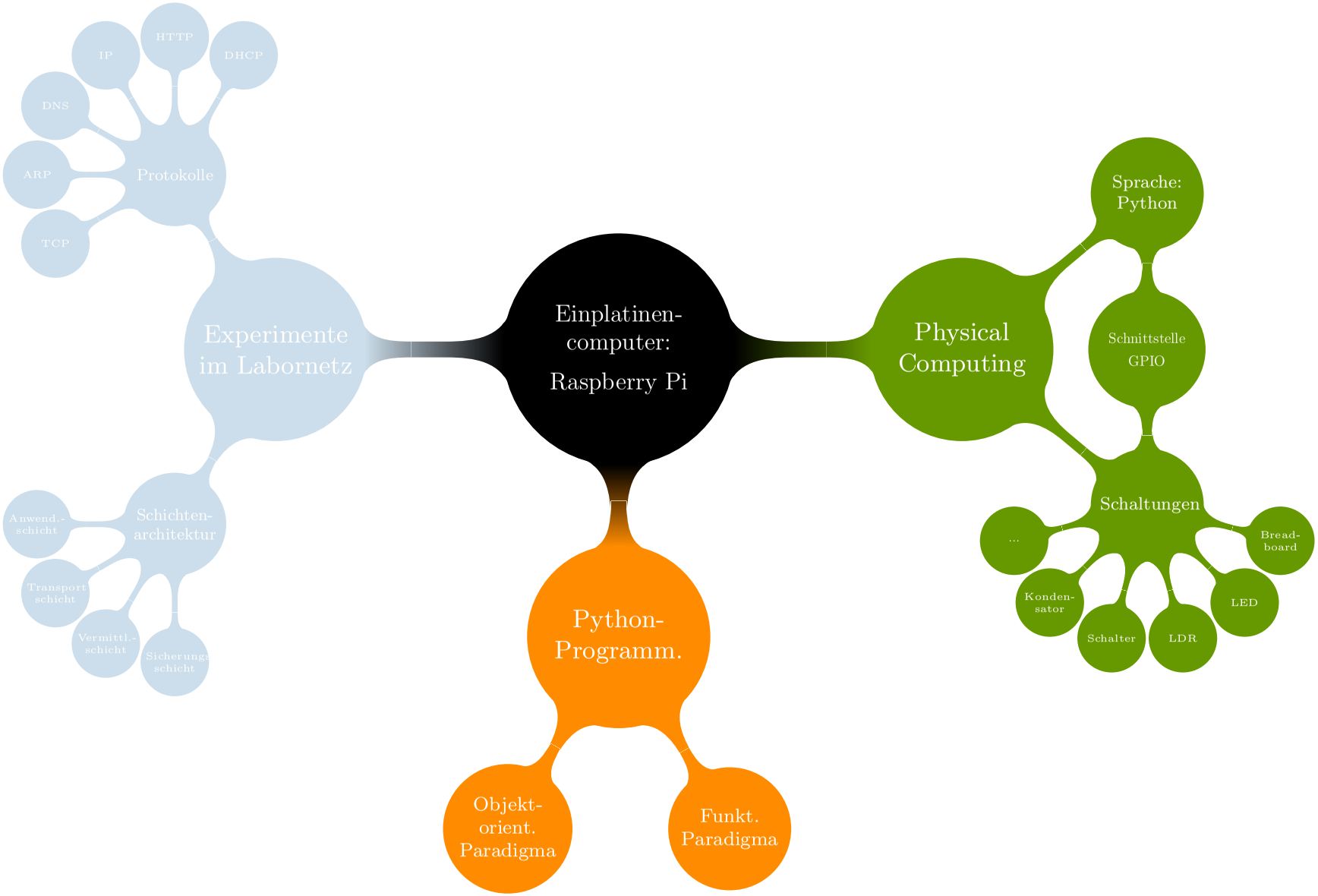


Abbildung 1.4: Zusammensetzung des Moduls: Rechnernetze (noch nicht umgesetzt), Python-Programmierung und Physical Computing

Kapitel 2

Hardware- und Software Anforderungen

Zunächst möchten wir auf die technischen Voraussetzungen eingehen. Wir unterscheiden dabei erforderliche Hardware und erforderliche Software:

2.1 Hardware

Jeder Arbeitsplatz benötigt einen Raspberry Pi und Zubehör. Wir haben im Folgenden das wichtigste Zubehör aufgelistet:

- Raspberry-Pi Schutzhülle
- Stromkabel USB auf Micro-USB
- HDMI zu VGA / DVI Adapter
- Maus und Tastatur
- SD-Karten (Festplatte)

Für die physikalischen Experimente benötigen wir noch einen Cobbler, Breadboard und diverse Bauteile. Eine Übersicht über die von uns verwendeten Bauteile finden Sie im Anhang.

2.2 Software

Wir haben eine Image-Datei erstellt, die Sie auf eine geeignete 8-GB-SD-Karte aufspielen können. Auf dieser SD-Karte befinden sich dann alle von uns verwendeten Programme.
Sie müssen sich somit nicht um die Installation unter Linux kümmern und benötigen folglich auch keinen Internetzugang für die Raspberry Pis.

Wir beschreiben die SD-Karten mit der Software Win32DiskImager:

http://www.chip.de/downloads/Win32-Disk-Imager_46121030.html

Kapitel 3

Programmierung mit Python

Überblick:

In Kapitel 4 beschäftigen wir uns mit Physical Computing. Wir werden dabei auch die GPIO-Pins des Raspberry Pi mittels Programmiersprache ansteuern, um z. Bsp. Spannung anlegen oder messen zu können. Dazu verwenden wir die Programmiersprache Python.
Um die Einstiegshürde für Kapitel 4, mit der vielen Schülerinnen und Schüler unbekannten Programmiersprache Python auf der einen Seite und den physikalischen Schaltelementen auf der anderen Seite, nicht zu hoch zu legen, wollen bieten wir in diesem Kapitel einen schrittweisen Einstieg in Python. Ziel ist es, die Schülerinnen und Schüler mit wichtigen Befehlen und der Programmiersyntax von Python vertraut zu machen.

3.1 Die Programmiersprache Python

In diesem Abschnitt gehen wir kurz auf Installation von Python und dem von uns verwendeten Editor ein. Für dieses Kapitel ist der Raspberry Pi nicht zwingend notwendig - Sie können die Aufgaben auf jedem Computer, auf dem Python installiert ist, bearbeiten.
Zudem verwenden wir für dieses Kapitel als Editor `IDLE (Python GUI)`, der direkt mit Python installiert wird.

3.1.1 Installation

3.1.1.1 Installation unter Windows

Man kann Python kostenlos von folgender Seite herunterladen:

http://www.python.org/download/releases/2.5/

Installieren Sie die heruntergeladene Datei.

3.1.1.2 Installation unter Linux (Raspberry Pi)

Auf dem Raspberry Pi ist standardmäßig Python installiert. Sie können in der Kommandozeilenumgebung (Terminal) via

```
1 python –V
```

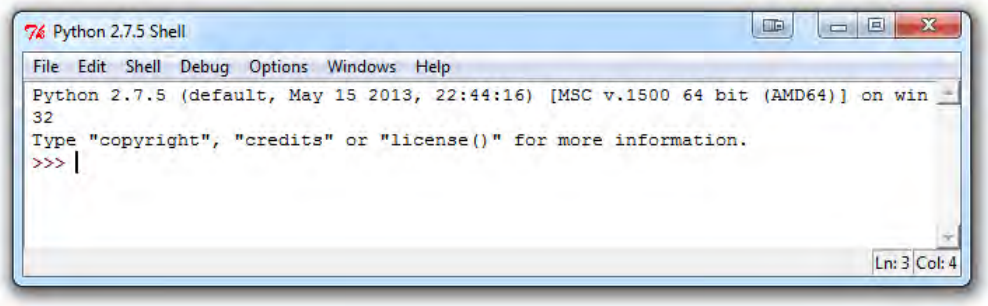
überprüfen, welche Version von Python installiert ist.

3.1.2 Dateiendungen

Python-Dateien müssen als Dateiendung `.py` besitzen, sonst lassen sie sich nicht ausführen. Ob man die Dateiendung vergessen hat, kann man schnell daran erkennen, dass die Schlüsselwörter von Python, wie etwa `import, if, else, while`, nicht farbig eingefärbt werden.

3.1.3 Erste Schritte mit Python

Öffnet man IDLE kann man prinzipiell sofort mit dem Programmieren loslegen. Anders wie bei Java benötigt man keine Klassen und eine `main`-Methode, um ein Programm ausführen zu können.
Es erscheint ein Prompt `>>>`, hinter den wir immer einen Befehl schreiben können:



Unterschiede der Schreibweise zu Java

Python unterscheidet sich in vielerlei Hinsicht von Java. Aus syntaktischer Sicht ist der größte Unterschied, dass in Python keine geschweiften Klammern und Strichpunkte verwendet werden. Dafür ist in Python die Einrückung sehr wichtig!
Betrachten wir hierzu einige Beispiele:

Da in Python die Klammern fehlen, ist hier die Einrückung im Gegensatz zu Java für die Bedeutung (Semantik) wichtig! Es muss also immer darauf geachtet werden, dass richtig eingerückt wird. Wichtig sind auch die

```python
if i > 5:
    n = n + i
    n = n * n
else:
    n = n - i

print(n)
```

```java
public void test() {

    if (i > 5) {
        n = n + i;
        n = n * n;
    } else {
        n = n - i;
    }

    System.out.println(n);
}
```

Doppelpunkte hinter `if`, `else`, etc. Viele Fehler beim Compilieren sind darauf zurückzuführen!

```python
def printName(name):
    print(name)


def summe(a,b):
    n = a + b
    return n
```

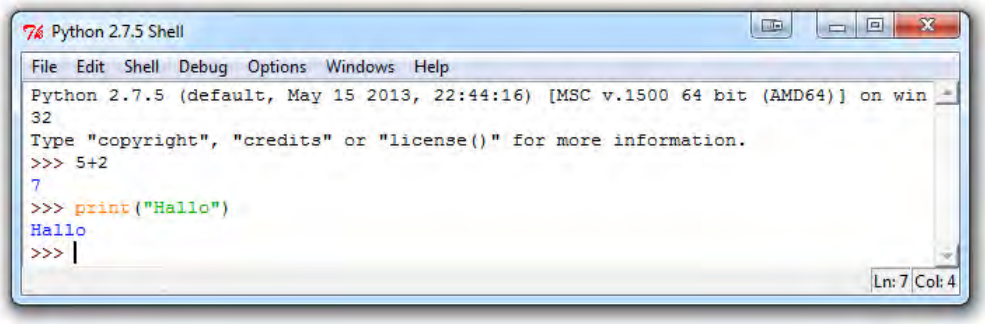
```java
public void printName(String name) {
    System.out.println(name);
}


public int summe(int a, int b) {
    int n = a + b;
    return n;
}
```

In Python kann man am Methodenkopf nicht erkennen, ob die Methode einen Rückgabewert hat oder nicht. Ebenfalls ist nicht erkennbar, welchen Rückgabetyp eine Methode hat, falls überhaupt einer vorhanden ist.

Im restlichen Abschnitt stellen wir noch wichtige Befehle und ihre Umsetzung in Python vor. Wir werden dabei immer Bezug auf ihre Umsetzung in Java nehmen. Die nachfolgenden Aufgaben sind so entworfen, dass die für die Bearbeitung wichtigen Befehle noch einmal vorgestellt werden.

3.1.3.1 Mathematische Operatoren und Textausgabe

Einfache Beispiele sind das Addieren von zwei Zahlen oder die Ausgabe einer Zeichenkette:



3.1.3.2 Eigenes Programmmodul erstellen

Möchten wir mehr programmieren, als einzelne Zeilen, ist es sinnvoll, das Programm in ein eigenes Fenster (Modul) zu schreiben, da wir sonst nach Programmausführung nichts mehr ändern können.
Dies geht über `File` → `New Window`. Dieses Modul lässt sich über `Run` → `Run Module` ausführen.[1]

3.1.3.3 Wenn ... dann ... sonst

Für die Syntax der Kontrollstruktur `if/else` betrachten wir folgendes Code-Beispiel:

```
1 if name == "Bob Marley":
2     shotTheSheriff = True
3 elif name == "Taio Cruz" or "Flo Rida":
4     hasHangover = True
5 elif name == "In Flames":
6     zuhoeren()
7 else:
8     print(name)
```

3.1.3.4 Logische Operatoren

Die logischen Operatoren unterscheiden sich von denen in Java. Lediglich der Vergleichsoperator `==` ist identisch. Betrachten wir dazu wieder ein äquivalentes Umsetzungsbeispiel in Python und Java:

```
def test(a,b,c,d):
    if (a==b) and (c==d):
        return True
    elif (a==b) or (c==d):
        return False
    else:
        return a
```

```
public boolean test(boolean a, boolean b, boolean c, boolean d) {
    if ((a == b) && (c == d)) {
        return true;
    } else if ((a == b) || (c == d)) {
        return false;
    } else {
        return a;
    }
}
```

Beachten Sie auch, dass `True` und `False` in Python mit einem Großbuchstaben beginnen.

[1] Lässt sich das Module nicht ausführen, kann das daran liegen, dass Sie `.py` als Dateiendung vergessen haben.

3.1.3.5 Eigene Methoden und Wiederholungen

In Abschnitt 3.1.3.4 haben Sie bereits gesehen, wie in Python eigene Methoden definiert werden können. In diesem Abschnitt ist die allgemeine Syntax und noch eine Beispielmethode aufgeführt:

```
# Programm-Code

def meineMethode(Parameter)
    # Methoden-Code

# Programm-Code
```

```
def geradeZahl(zahl):
    if zahl % 2 == 0:
        print("Gerade Zahl")
    else:
        print("Ungerade Zahl")
```

Bei den Wiederholungen ähnelt `while` sehr der Umsetzung in Java:

```
while i < 5:
    # innerer Rumpf der while-Schleife
```

```
while (i < 5) {
    // innerer Rumpf der while-Schleife
}
```

Bei der Wiederholung mit festen Anzahl (`for`) gibt es allerdings Unterschiede:

```
for i in range(6):
    # i nimmt nacheinander
    # die Werte 0,1,2,3,4,5 an
```

```
for (int i = 0; i < 6; i++) {
    // i nimmt nacheinander
    // die Werte 0,1,2,3,4,5 an
}
```

3.1.3.6 Klasse, Konstruktor und Klassenmethoden

Eine Klasse in Python kann beispielsweise wie folgt ausschauen:

```
# Klassenname
class Auto():

    # Konstruktor - saemtliche Klassenvariablen muessen so im
    # Konstruktor aufgefuehrt sein
    def __init__(self,farbe,laenge):
        self.farbe = farbe
        self.laenge = laenge
        self.breite = laenge/3

    # Klassenmethoden brauchen immer self als ersten Parameter
    # Dieser wird beim Aufruf der Methode mit keinem Wert belegt
    def farbewechseln(self,neueFarbe):
        self.farbe = neueFarbe

    def gibFarbe(self):
        farbe = self.farbe
        return farbe
```

Betrachten wir die Bestandteile der Klasse genauer. Die Klasse beginnt mit den Schlüsselwörtern

```
class Klassenname():
```

Die erste Methode, die man in jeder Klasse braucht, ist der aus Java bekannte Konstruktor. Seine Kopfzeile schaut wie folgt aus

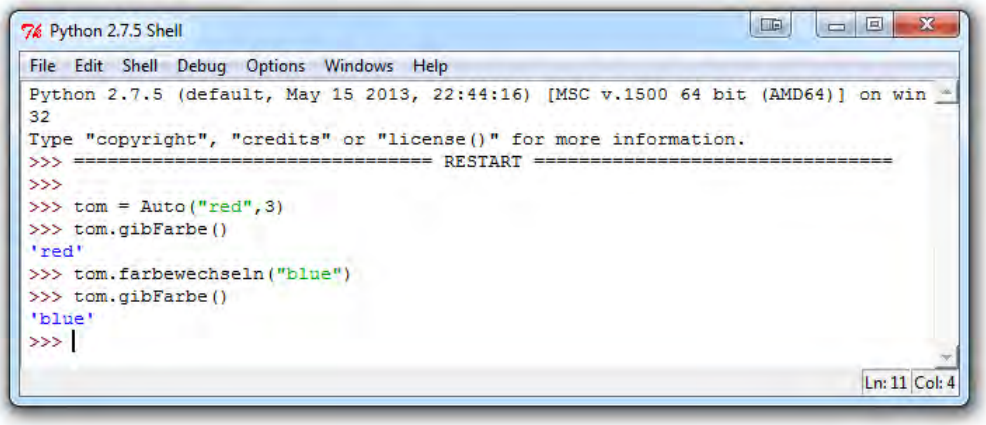
```
def __init__(self,attributname1,...):
```

Im Konstruktor kommen meist Anweisungen wie

```
self.attributname1 = attributname1
...
self.neueVariable1 = wert
...
```

Jede weitere Methode braucht als erste Übergabevariable `self`. Daran erkennt die Methode, dass sie zum Objekt ihrer Klasse gehört. Eine Ausführung der Beispiel-Klasse `Auto.py` könnte wie folgt aussehen:

Zunächst muss die Klasse als Modul geladen werden: `Run` → `Run Module`. Es erscheint `RESTART` in der IDLE. Danach erzeugen wir ein Objekt `tom` der Klasse und rufen Methoden auf.



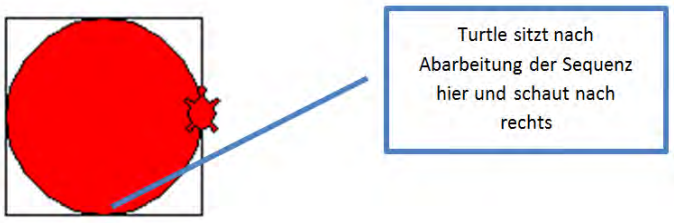
Die „Variable“ `self` aus der Definition der Methoden fällt unter den Tisch. Vergisst man, `self` als ersten Parameter bei der Definition der Methoden anzugeben, wird IDLE dies reklamieren.

3.2 Aufgaben

3.2.1 Sequenz und Wiederholung

Beispiel 1:

```
1  # -*- coding: utf-8 -*-
2  # Wenn man z.B. in Kommentaren Umlaute verwenden moechte, muss eine Kodierung angegeben
       werden, mit der das moeglich ist. Diese Angabe muss am Anfang der Datei stehen
3
4  # Turtle-Modul importieren
5  import turtle
6  # Standardsymbol fuer Turtle setzen
7  turtle.shape("turtle")
8
9  # Quadrat zeichnen
10 turtle.forward(100)
11 turtle.left(90)
12 turtle.forward(100)
13 turtle.left(90)
14 turtle.forward(100)
15 turtle.left(90)
16 turtle.forward(100)
17 turtle.left(90)
18
19 turtle.forward(50)
20 # Fuellfarbe auf rot setzen
21 turtle.begin_fill()
22 turtle.fillcolor("red")
23
24 # Kreis in Quadrat setzen
25 turtle.circle(50)
26 turtle.end_fill()
```



Was tun, wenn das Zeichenfenster nach Abarbeitung des Skripts schließt?
Busy Waiting ans Ende des Skripts setzen - kann mit `Strg+C` beendet werden:

```
1 while True:
2     pass  # Platzhalter fuer leeren Rumpf
```

3.2.1.1 Wiederholung mit fester Anzahl

(a) Erstelle eine Datei `Quadrat.py`. Zeichne ein Quadrat und verwende dabei eine Wiederholung mit fester Anzahl (Syntax siehe Beispiel 2).
Orientiere dich an Beispiel 1. Dort sieht man, was zu Beginn eines Skripts importiert werden muss, wenn man mit einer Schildkröte arbeiten möchte und mit welchen Befehlen man die Schildkröte steuern kann.

(b) Experimentiere mit folgenden Methoden:

```
turtle.shape("circle")  # Turtle wird als Kreis dargestellt
turtle.fillcolor("red")
turtle.shapesize(5,1,4) # Breite, Hoehe, Dicke der Umrandungslinie --> Ellipse
turtle.tilt(10)         # Turtle um eigene Achse drehen, Drehwinkel = 10 Grad
turtle.stamp()          # Turtle hinterlaesst einen Abdruck
```

Die Schildkröte verwendet ihren Zeichenstift nicht. Sie hinterlässt an jeder Position ein Abbild und bewegt sich vorwärts und dreht sich dabei zusätzlich um die eigene Achse. Dabei können nette Figuren entstehen.

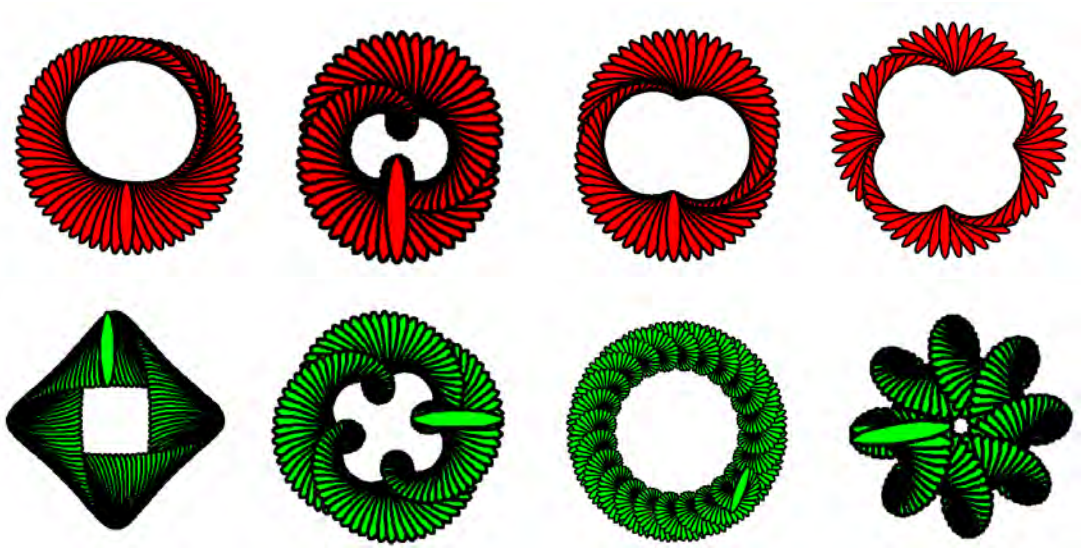


Abbildung 3.1: Figuren, die durch Experimentieren mit den Befehlen `turtle.forward(..)`, `turtle.left(..)`, `turtle.tilt(..)`, `turtle.stamp()` entstanden sind.

Hinweis: Ein Vollwinkel hat 360°. Wenn sich die Schildkröte beispielsweise um 5° dreht (`turtle.left(5)`), dann solltest du dies 72 mal iterieren (wiederholen), damit die Schildkröte wieder im Ausgangszustand ist.

3.2.2 Eigene Methoden definieren

Eine Methode wird in Python wie folgt festgelegt:

```
def methodenname(ggf. parameternamen):
    # Methodenrumpf
```

Nach dem Doppelpunkt kommt dann, *eingerückt in einer neuen Zeile*, eine Sequenz von Anweisungen.

Beispiel 2:

```
1 def meinPolygonzug(seitenlaenge,winkel):
2     turtle.pensize(10)
3     turtle.color("red")
4     for i in range(3):
5         turtle.forward(seitenlaenge)
6         turtle.left(winkel)
```

3.2.2.1 Bilderrahmen

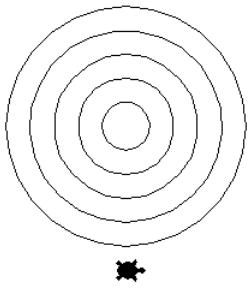
Erstelle eine neue Datei `Bilderrahmen.py`.

Definiere innerhalb dieser Datei eine Methode, die einen „Bilderrahmen“ der Größe 200×100 und Dicke 10 zeichnen lässt.

Teste deine Methode!

3.2.2.2 Mehrere Kreise umeinander

Schreibe eine Methode, die erst einen kleinen Kreis zeichnet und anschließend noch vier Kreise darum herum, so dass das Bild wie folgt ausschaut:



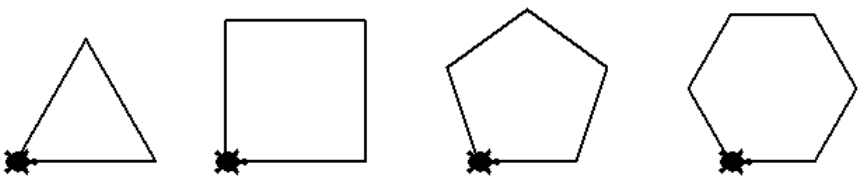
Hinweis:

```
1 turtle.penup()      # Zeichenstift anheben
2 turtle.pendown()    # Zeichenstift absenken
3 turtle.circle(r)    # Kreis mit Radius r zeichnen
```

Du kannst dich etwas an Beispiel 1 orientieren. Verwende wieder eine Wiederholung mit fester Anzahl!

3.2.2.3 n-Eck

Schreibe eine Methode, die eine Schildkröte ein regelmäßiges n-Eck zeichnen lässt!



Hinweis: Für ein Dreieck gilt: Wenn die Schildkröte einmal um ein **Dreieck** herumgelaufen ist, dann hat sie eine 360° Drehung hinter sich. An jeder Ecke muss sie also **ein Drittel** dieses Winkels überstreichen!

```
1 turtle.left(120)
```

3.2.3 Methoden (bzw. Funktionen) können als Argumente von Methoden (bzw. Funktionen) vorkommen

Beispiel 3 (ohne Schildkröte):

```
1  from math import cos, sin
2
3  def funktion(adjektiv):
4      return "Ich bin eine einfache, aber " + adjektiv + " Funktion."
5
6  # f muss eine Funktion mit einem Parameter sein
7  # "der Typ von x muss zu f passen"
8  def drucken(f, x):
9      # str - umwandeln in eine Zeichenkette
10     print("The function passed to me says: " + str(f(x)))
11
12 drucken(funktion, "schoene")
13 drucken(cos, 0)
14 drucken(cos, "schoene") # TypeError: a float is required
```

3.2.3.1 Drucken

Setze eine nette kleine Funktion ohne Parameter und eine mit zwei Parametern um. Schreibe dazu passend eine Methode `drucken0(..)` und eine Methode `drucken2(..)`. Orientiere dich dabei an Beispiel 3 und teste deine Umsetzung!

3.2.4 Funktionen können als Werte von Funktionen vorkommen

Beispiel 4:

```
1  from math import cos, sin
2
3  # f muss eine Funktion mit einem formalen Parameter sein
4  def ableitung_von(f):
5      dx = 1.e-6  # 1 mal 10 hoch -6 (ganz kleiner Wert)
6      def d_nach_dx_von_Argument(x):
7          return (f(x+dx)-f(x))/dx    # es wird der Differenzenquotient zurueckgegeben
8      return d_nach_dx_von_Argument
9
10 print(ableitung_von(sin)(0))
```

Inspiriert von Gregor Lingl: Seven Ways to use Python's new Turtle Module

3.2.4.1 Ableitungen

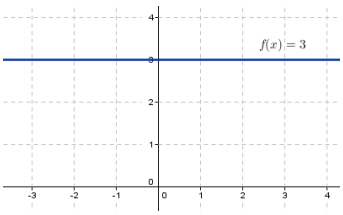
(a) Definiere zu Übungszwecken selbst eine Funktion `quadrat(x)`. Leite diese mit Hilfe des obigen Beispiels (näherungsweise) an einigen Stellen ab.

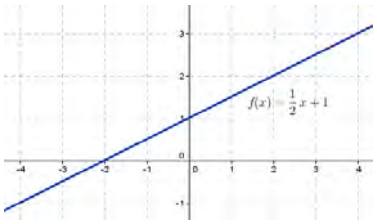
(b) Definiere eine Funktion `zweite_Ableitung_von`, welche (näherungsweise) die zweite Ableitung bestimmt. Orientiere dich dabei an obigen Beispiel und verwende dieses!

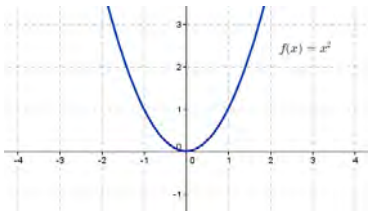
Exkurs: Ableitung, Differenzenquotient, Differentialquotient

Möglicherweise sagen dir die Begriffe Ableitung, Differenzenquotient, Differentialquotient (noch) nichts. Deswegen möchten wir dir die Begriffe im Folgenden kurz erklären und etwas näher bringen.

Betrachte zunächst folgende drei Funktionen:

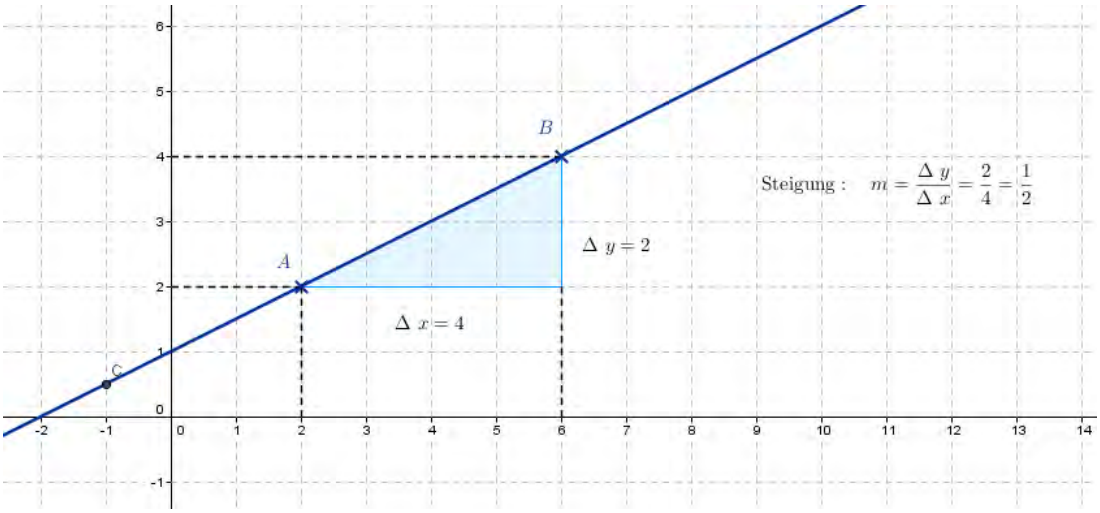






Welche Steigung haben diese drei Funktionen? Diese Frage kannst du für die linken beiden Funktionen sicherlich schnell beantworten, die rechte Funktion sollte dir größere Schwierigkeiten bereiten.
Anders als bei den zwei linken Funktionen, ändert sich die Steigung bei der rechten Funktion in jedem Punkt; z. Bsp. hat man in $(0, 0)$ Steigung 0 und in $(1, 1)$ Steigung 2.

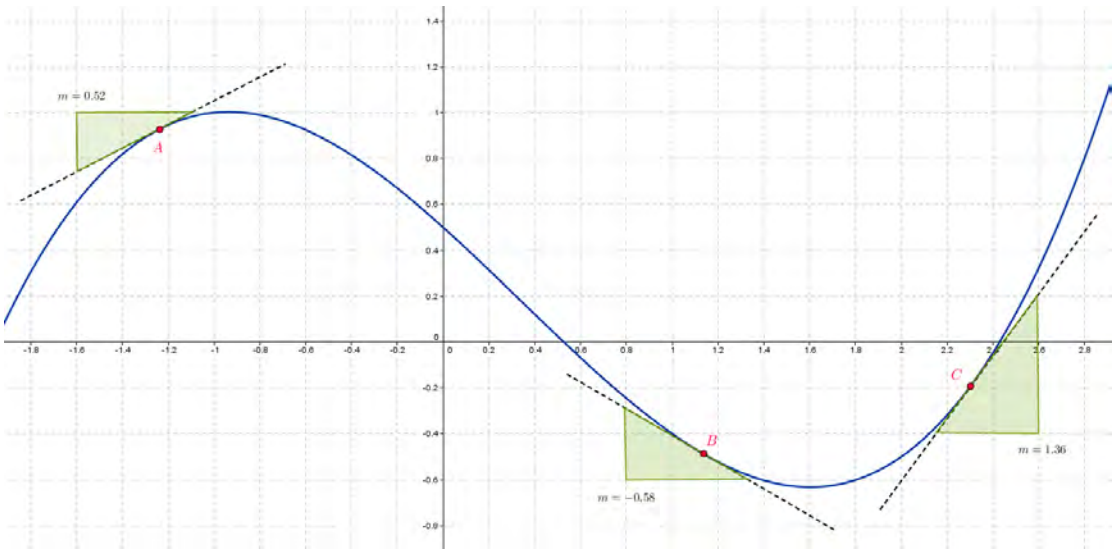
Die Steigung bei linearen Funktionen kann man schnell über das Steigungsdreieck berechnen:



Diese Eigenschaft nutzt man aus, um auch die Steigungen bei krummlinigen Graphen berechnen zu können. Man definiert die Steigung wie folgt:

> Die Steigung eines Funktionsgraphen in einem Punkt P ist gleich der Steigung der Tangente an den Graphen in diesem Punkt.

Betrachte dazu folgende Funktion:

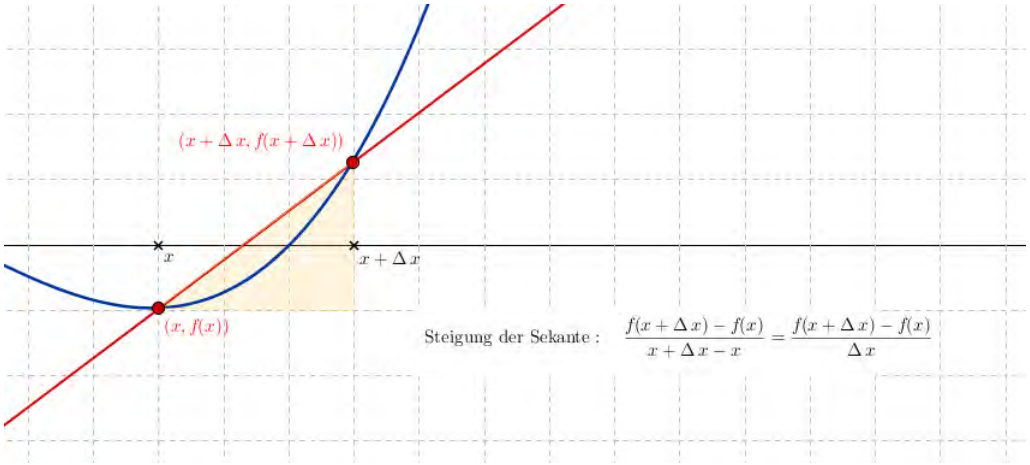


Wir haben nun die Steigung eines beliebigen Graphen in einem Punkt x über die Steigung der Tangente an den Graphen in diesem Punkt definiert. Wie lässt sich nun diese Tangentensteigung berechnen?

Hier kommt nun der Differenzenquotient ins Spiel:

$$\frac{f(x+\Delta x)-f(x)}{\Delta x}$$

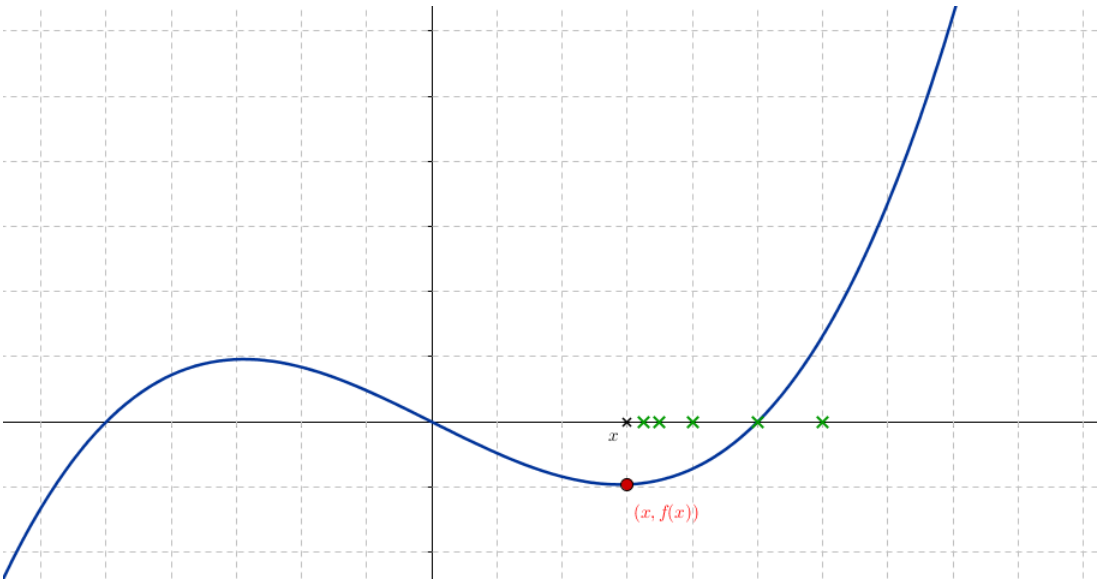
Dieser Quotient ist nichts anders, wie die Steigung der Gerade durch die Punkte $(x, f(x))$ und $(x+\Delta x, f(x+\Delta x))$ - Δx ist dabei eine beliebige reelle Zahl. Betrachten wir diese Situation grafisch:



Wie hilft uns der Differenzenquotient, die Steigung der Tangente im Punkt $(x, f(x))$ zu berechnen?

Das sollst du nun selber herausfinden!

Zeichne dazu in nachfolgender Grafik für unterschiedliche Werte von Δx (dargestellt mit den grünen Kreuzen) die Punkte $(x+\Delta x, f(x+\Delta x))$ ein. Zeichne zudem die Tangente an den Graphen im Punkt $(x, f(x))$ ein! Was fällt dir auf?



Der Mathematiker spricht hier vom *Grenzübergang* $\Delta x \to 0$ (Notation: $\lim_{\Delta x \to 0}$) und definiert die Steigung in einem Punkt $(x, f(x))$ als

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Dieser Ausdruck wird auch *Differenzialquotient* oder *Wert der ersten Ableitung* im Punkt $(x, f(x))$ genannt.

(c) Zeichne mit Hilfe der Schildkröte einige Funktionsgraphen

```
1  import turtle
2  from turtle import Screen
3
4  def graph(f):
5      screen = Screen()                        # Singleton-Objekt
6      screen.setworldcoordinates(-4,-2,4,2) # Skalierung fuer Achsen festlegen
7      x = -4
8      turtle.penup()
9      turtle.goto(x,f(x))
10     turtle.pendown()
11
12     while x < 4:
13         x = x+0.1
14         turtle.goto(x,f(x))
```

Inspiriert von Gregor Lingl: Seven Ways to use Python's new Turtle Module

Schreibe eine Methode `ablgraph(f)`, die dir zu einer Funktion den Graphen der Ableitungsfunktion zeichnet, d.h. den Graphen der Funktion, die jedem Punkt x den Funktionswert $f'(x)$ zuordnet.
Verwende die Funktion `ableitung_von` aus Beispiel 4, um den Wert von $f'(x)$ zu berechnen. Du kannst dich an der vorgegebenen Methode `graph(f)` orientieren.

3.2.5 Ereignisse und bedingte Anweisung

3.2.5.1 Schildkröte mit der Maustaste drehen

Beispiel 5:

```
1  import turtle
2
3  # Methode mit zwei formalen Parametern - kann als Argument fuer die vordefinierte Methode
       onclick verwendet werden
4  def drehen(x,y):
5      print(x)    # Mal schauen, welchen Wert x annimmt, wenn man klickt
6      turtle.left(90)
7
8  turtle.shape("turtle")
9  turtle.shapesize(3,3,3) # Schildkroete vergroessern
10 turtle.onclick(drehen)  # Die Schildkroete soll sich drehen, wenn man sie anklickt
11
12 turtle.mainloop()   # Endlose Wiederholung zwecks Ereignisverarbeitung anschalten
```

Ausprobieren!

Erweitere die Methode `drehen(x,y)` so, dass man durch Klicken auf den rechten Teil der Schildkröte diese nach rechts dreht, sonst nach links.

Beispiel 6:

```
1  if x < 0:
2      x = 0
3      print("Negative changed to zero")
4  elif x == 0:
5      print("Zero")
6  elif x == 1:
7      print("Single")
8  else:
9      print("More")
```

Via `turtle.heading()` kann man herausfinden, in welche Richtung die Schildkröte schaut. Je nach Blickrichtung der Schildkröte befindet sich deren rechte Seite tatsächlich „rechts“ oder „oben“ oder „links“ oder „unten“.

Hinweis: Es gibt in Python die boolschen Operatoren `and`, `or` und `not`

Sollte eine Zeile im Programmcode einmal zu lang, werden, so kann man einen Backslash \einfügen und die Zeile danach umbrechen. Dann wird der Zeilenumbruch nach dem \ nicht als Teil einer Zeichenfolge interpretiert, die einen neuen Block beginnt

3.2.5.2 Schildkröte mit zwei Farbzuständen

Nun soll sich die Schildkröte nicht mehr drehen, wenn man auf sie „drückt“, sondern ihre Farbe wechseln. Vielleicht von schwarz nach rot und zurück.
Via `turtle.fillcolor()` lässt sich herausfinden, welche Füllfarbe die Schildkröte gerade hat.

`turtle.color("red")` setzt sowohl Füllfarbe als auch Umrandung der Kröte auf rot. Schwarz gibt es auch als vordefinierte Farbe: `"black"`.

3.2.5.3 Schildkröte folgt dem Mauszeiger und zeichnet

Zu Beginn des Skripts musst du die Klasse Screen importieren, dann ein Screen-Objekt erzeugen und mit diesem arbeiten.

Informiere dich über die `onclick`-Methode[2], die auch von einem Screen-Objekt angeboten wird. Man kann zwei Argumente an die Methode übergeben. Hebe mit einem Rechtsklick den Zeichenstift an und senke den Zeichenstift mit einem erneuten Rechtsklick.
Via `turtle.isDown()` lässt sich herausfinden, ob der Zeichenstift gerade abgesenkt ist oder nicht. Wenn du links klickst, soll die Maus ihrem Mauszeiger folgen; hier kannst du dich an Beispiel 7 orientieren.

Beispiel 7:

```
1 import turtle
2 from turtle import Screen
3
4 screen = Screen()
5 screen.onclick(turtle.goto) # goto ist eine vordefinierte Funktion mit zwei Parametern x und y
6
7 turtle.mainloop()
```

Hinweis: Den Maustasten sind Zahlen zugewiesen; dabei steht 1 für die linke, 2 für die mittlere und 3 für die rechte Maustaste.

`onclick(fun, btn=1)`

`fun`: Funktion mit 2 Argumenten

`btn`: 1 oder 2 oder 3 für Maustasten

Bindet den Aufruf von `fun` an das Mausklick-Ereignis ins Grafik-Fenster mit der entsprechenden Maustaste. Die Koordinaten des angeklickten Punktes werden beim Aufruf an `fun` als Argumente übergeben.

[2] Falls du kein Internet zur Verfügung hast, kannst du dich auch am unten stehenden Hinweis orientieren

3.2.6 Erstellen einer eigenen Klasse

3.2.6.1 Die Klasse Vieleck

Bei dieser Aufgabe wollen wir eine Klasse Vieleck erstellen, mit der man regelmäßige Vieleck-Objekte erstellen kann. Zudem soll die Klasse verschiedene Methoden anbieten - z. Bsp. das Einzeichnen des Mittelpunkts des Vielecks.

Wir haben dir bereits den Rumpf der Klasse zur Verfügung gestellt:

```
1  import math
2  from turtle import Turtle
3
4  # Klasse Vieleck
5  class Vieleck():
6      #Konstruktor, der die Anzahl der Ecken und die Seitenlaenge erwartet
7      def __init__(self,ecken,slaenge):
8          self.ecken = ecken
9          self.laenge = slaenge
10         self.turtle = Turtle(shape="turtle")
11         self.turtle.pensize(2)
12
13     def zeichenWinkel(self):
14         # hier soll der Zeichenwinkel berechnet werden
15
16     def zeichnen(self):
17         # hier soll das regelmaessige Viereck gezeichnet werden
18         # verwende dazu das Attribut self.turtle!
```

Lese dir vor Bearbeitung dieser Aufgabe den Einführungsabschnitt 3.1.3.6 zu Klassen, Konstruktor und Klassenmethoden durch.

Ersetze nun die Kommentare in den Methoden `zeichenWinkel(self)` und `zeichnen(self)` durch Python-Code. Die Methode `zeichenWinkel(self)` soll dabei den Zeichenwinkel zurückgeben. Als Zeichenwinkel bezeichnen wir den Winkel, um den sich die Schildkröte an jedem Eck des regelmäßigen n-Ecks drehen muss. Wir haben dir diesen in Abbildung 3.2 für ein Fünf- und Sechseck eingezeichnet.

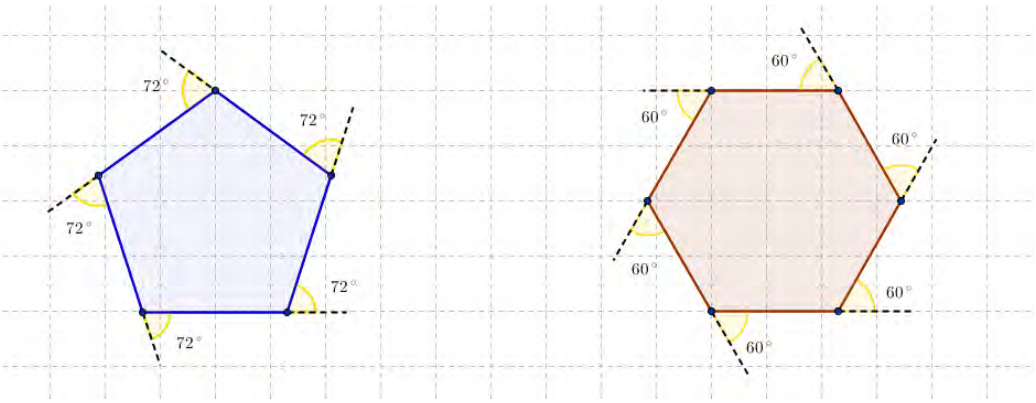
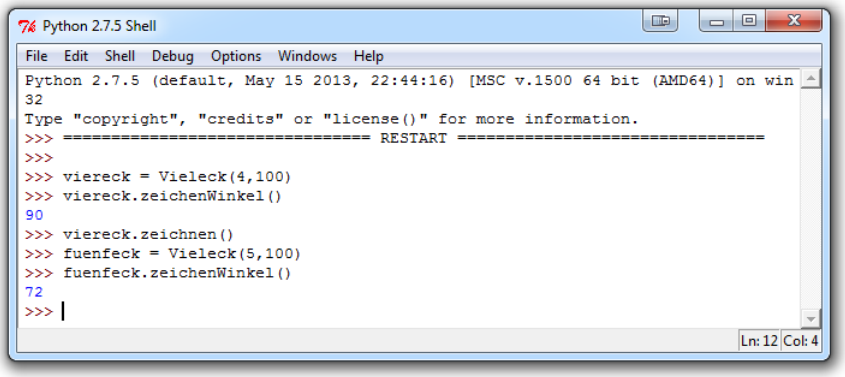
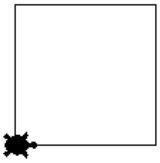


Abbildung 3.2: Zeichenwinkel bei einem regelmäßigen Fünf- bzw. Sechseck

Teste deine Klasse und ihre zwei Methoden:



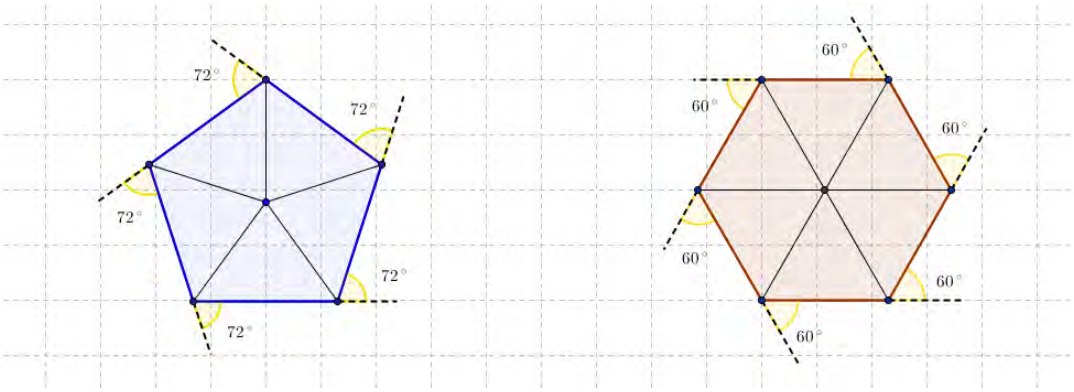


Mittelpunkt einzeichnen

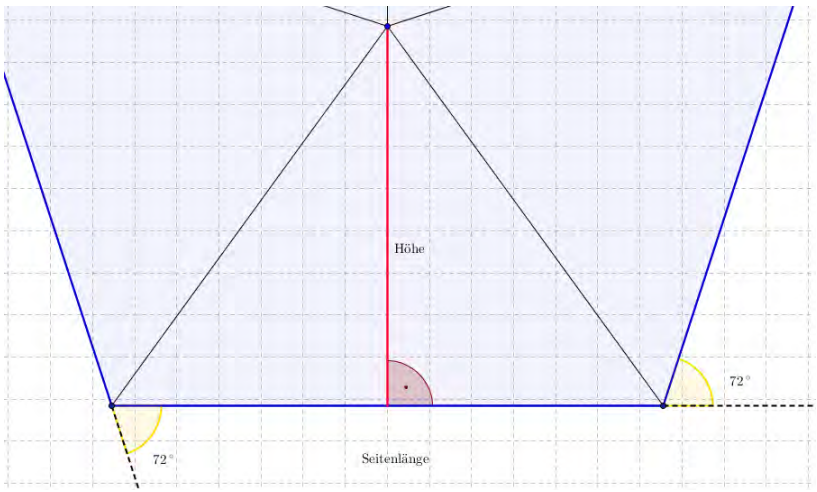
Nachdem das regelmäßige n-Eck gezeichnet wurde, soll nun eine Methode angeboten werden, die den Mittelpunkt des Vielecks einzeichnet. Wir gehen dabei davon aus, dass die Schildkröte nach dem Zeichnen links unten im Vieleck sitzt - vgl. Aufgabe 3.2.2.3. Ändere ggf. deine `zeichnen()`-Methode dahingehend ab!!

Für diese Aufgabe braucht man nun etwas Mathematik.

Alle regelmäßigen n-Ecke haben gemeinsam, dass man ihnen n gleichschenklige Dreiecke einbeschreiben kann. Exemplarisch für Fünf- bzw. Sechseck:

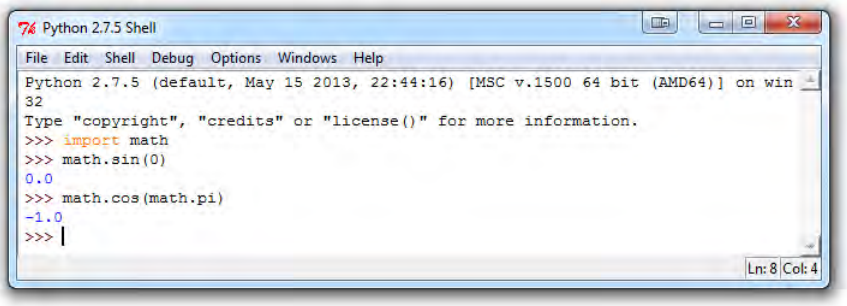


Betrachten wir ein solches gleichschenkliges Dreieck näher:



Kannst du aus gegebenen Werten die Höhe berechnen?

Hinweis: Python bietet die trigonometrischen Funktionen `sin, cos` und `tan` an.



Alle Funktionen erwarten in Python als Argument einen Winkel **im Bogenmaß**. Einen Winkel α, der in Grad gegeben ist, rechnest du wie folgt in Bogenmaß um:

$$\alpha \cdot \frac{\pi}{180}$$

Vervollständige nun folgende Methoden und füge sie deiner Klasse `Vieleck.py` hinzu:

```
1 def berechneHoehe(self):
2     # diese Methode soll die Hoehe der gleichschenkligen Dreiecke berechnen und zurueckgeben
3
4 def zeichneMittelpunkt(self):
5     # diese Methode soll den Mittelpunkt des gezeichneten Vielecks einzeichnen - dabei ist es
          hilfreich die Methode berechneHoehe() zu verwenden!
```

Teste deine zwei neuen Methoden:

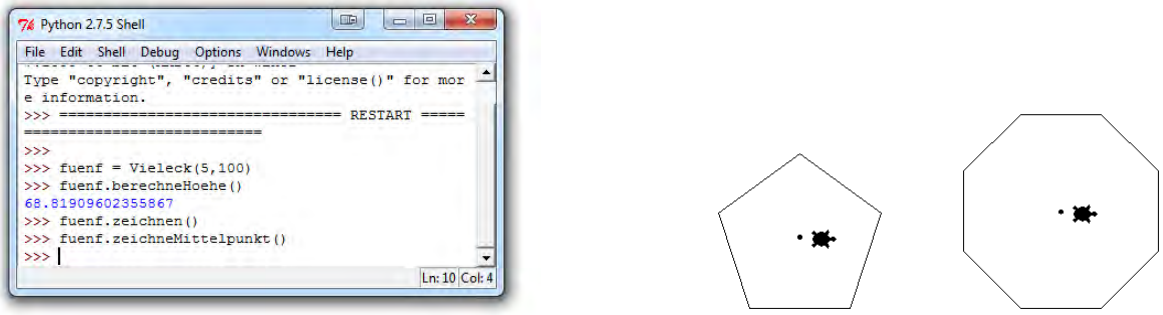
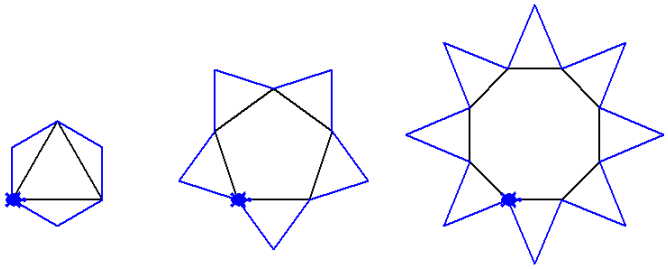


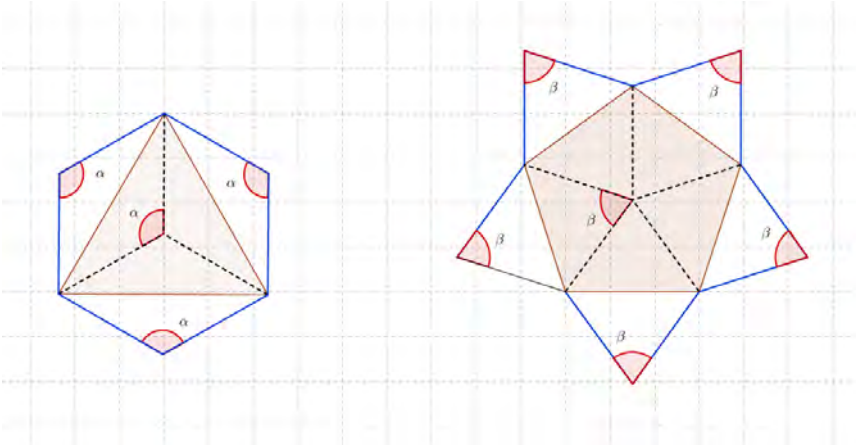
Abbildung 3.3: Links die Befehle, um die Methode `zeichneMittelpunkt()` bei einem Fünfeck zu testen. Exemplarisch dazu auch noch das Ergebnis für ein Achteck.

Stern zeichnen

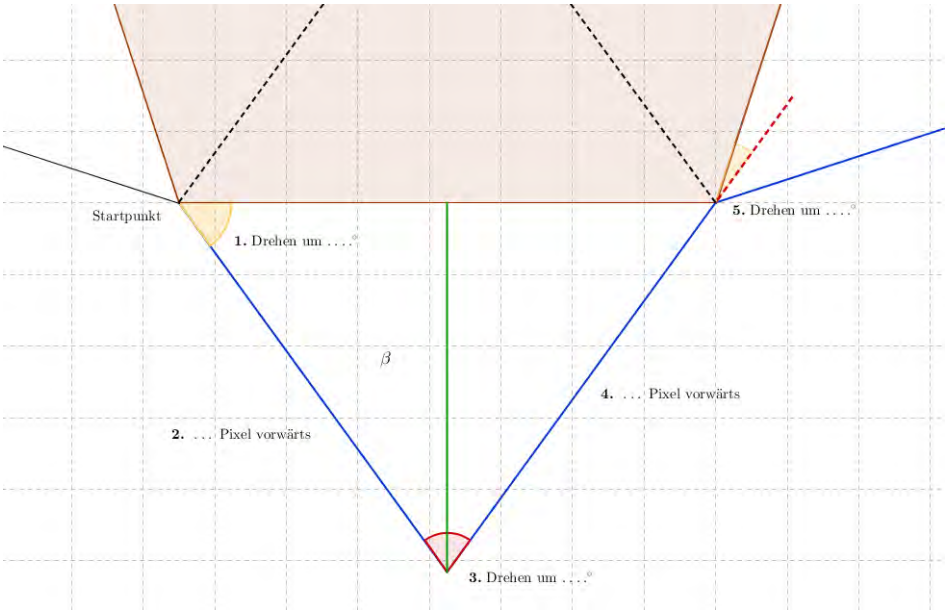
Zum Abschluss dieser Aufgabe sollst du noch eine Methode schreiben, die aus einem regelmäßigen Vieleck einen „Stern“ macht, indem es an den Seiten gleichschenklige Dreiecke anfügt. Dies kann wie folgt aussehen:



Die Methode `zeichneStern()` soll so implementiert werden, dass der Winkel in der Sternspitze der Mittelpunktswinkel des Vielecks sein soll. Du kannst dich dabei an nachfolgendem Bild orientieren:



Du kannst annehmen, dass deine Schildkröte wieder in der linken unteren Ecke startet. Überlege dir anhand einer Skizze, wann sich die Schildkröte um wie viel Grad drehen und wann sie sich um wie viele Pixel nach vorne bewegen muss!
Nachfolgender Ausschnitt kann dir bei deinen Überlegungen helfen:



Vervollständige nun folgende Methoden und füge sie deiner Klasse `Vieleck.py` hinzu:

```
def berechneDrehwinkel1(self):
    # diese Methode soll den Drehwinkel von Punkt 1 berechnen

def berechneWeglaenge2(self):
    # diese Methode soll die Weglaenge von Punkt 2 berechnen

def berechneDrehwinkel3(self):
    # diese Methode soll den Drehwinkel von Punkt 3 berechnen

def berechneDrehwinkel5(self):
    # diese Methode soll den Drehwinkel von Punkt 5 berechnen

def zeichneStern(self):
    # diese Methode soll einen Stern mit oben genannten Eigenschaften um das Vieleck zeichnen
```

Beachte hierzu den Hinweis auf Seite 3.2.6.1 zu der Umsetzung trigonometrischer Funktionen in Python! Die Funktionen $\texttt{sin}^{-1}$, $\texttt{cos}^{-1}$ und $\texttt{tan}^{-1}$ heißen in Python `math.asin, math.acos` und `math.atan`.
Eine Zahl quadriert man in Python wie folgt: `x**2`

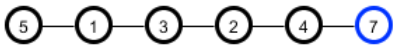
3.2.7 Listen in Python

Bei dieser Aufgabe beschäftigen wir uns mit Listen und ihrer Umsetzung in Python. Eine Liste ist eine sequentieller Datenstruktur, die sehr flexibel anwendbar ist. Sie bietet verschiedene vordefinierte Methoden, wie z. Bsp. das Anfügen neuer Elemente, an, die wir im folgenden näher betrachten werden.


```
>>> liste = [5,1,3,2,4]
>>> print liste
[5, 1, 3, 2, 4]
>>>
```

Hinzufügen und Löschen von Listenelementen

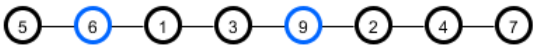
Wir arbeiten mit oben abgebildeter Liste. Mit dem Befehl `liste.append(x)` fügt man ein neues Element x am Ende der Liste ein. So führt der Befehl `liste.append(7)` zu folgender neuer Liste:



Weiter gibt es den Befehl `liste.insert(i,x)`, welche ein neues Element x an der i-ten Position der Liste einführt.

Beachte, dass bei Listen, wie auch bei Arrays in Java, immer bei 0 begonnen wird zu zählen. So steht in oben abgebildeter Liste das Element 5 an nullter Stelle, das Element 2 an dritter Stelle.

So führen die Befehle `liste.insert(3,9)` und `liste.insert(1,6)` zu folgender neuer Liste:



Entsteht dieselbe Liste, wenn man die Befehle vertauscht?

Der Befehl `len(liste)` gibt die Anzahl der Einträge der Liste zurück.

```
>>> print liste
[5, 6, 1, 3, 9, 2, 4, 7]
>>> len(liste)
8
>>>
```

Der Befehl `liste.pop(i)` hat das i-te Element der Liste als Rückgabewert und löscht dieses aus der Liste. So führt `liste.pop(3)` zu folgender Liste[3]:

3.2.7.1 Listen verschmelzen

Schreibe nun eine Methode, die zwei Listen als Parameter bekommt. Dabei soll die zweite Liste mit der ersten Liste verschmolzen werden, d.h. in der neuen Liste (kann in der Variable der ersten Liste gespeichert werden)

[3] Lässt man den Parameter i weg, wird immer das letzte Element aus der Liste entfernt

sollen sich die Elemente der ersten Liste und die Elemente der zweiten Liste befinden:

```
>>> liste1 = [1,2,3]
>>> liste2 = [4,5,6]
>>> verschmelzeListen(liste1,liste2)
>>> print liste1
[1, 2, 3, 4, 5, 6]
```

3.2.7.2 Die filter- und map-Funktion

Es gibt zwei vordefinierte Funktionen in Python, die sehr nützlich sind, wenn man sie zusammen mit Listen verwendet: `filter()` und `map()`.

Die Funktion `filter(funktion,liste)` gibt eine Liste zurück, die nur noch aus den Elementen `element` des Parameters `liste` besteht, für die `function(element)` `True` ergibt:

```
>>> filter(istgerade,range(2,15))
[2, 4, 6, 8, 10, 12, 14]
```

`range(2:15)` erzeugt dabei eine Liste mit allen Werten von 2 bis 15.

```
1 def durchDreiTeilbar(zahl):
2     # gibt True zurueck, falls durch drei teilbar - sonst False
```

Vervollständige oben angegebene Funktion und wende die `filter`-Funktion auf die Liste der Zahlen von 1 bis 30 an!

Die Funktion `map(funktion,liste)` wendet die Funktion `funktion` auf jedes Listenelement der Liste `liste` an und gibt die daraus entstandene Liste zurück. Z. Bsp. könnte man mit der `map`-Funktion alle Elemente einer Liste quadrieren:

```
>>> map(quadrieren,range(1,10))
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Erstelle ein Python-Programm, dass dir die Kubik-Zahlen von 1 bis 15 in einer Liste ausgibt!

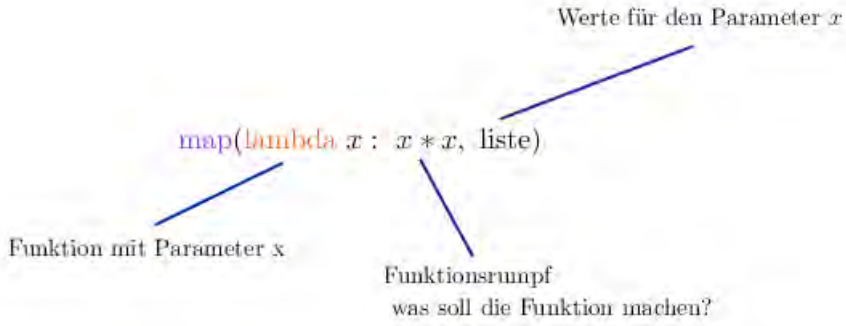
Möchte man nur „einfache“ Funktionen für den `map`-Befehl nutzen, geht dies oftmals noch schneller mit der λ-Funktion. Betrachten wir zunächst drei einfache Beispiele:

```
>>> liste = range(10)
>>> print liste
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> map(lambda x: x*x*x, liste)
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

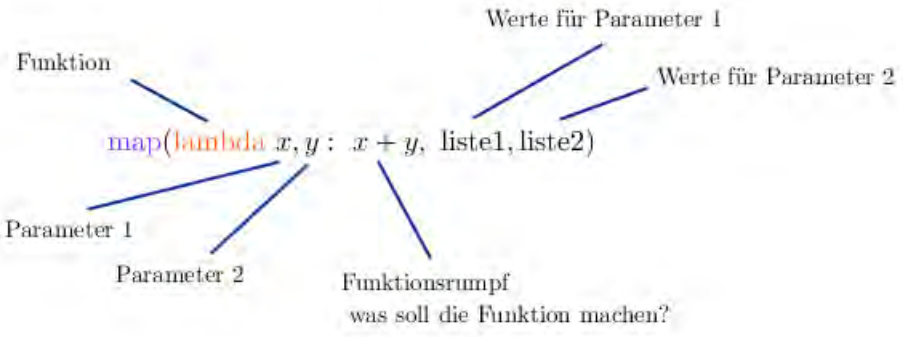
```
>>> liste1 = [1,2,3]
>>> liste2 = [4,5,6]
>>> map(lambda x,y: (x,y), liste1,liste2)
[(1, 4), (2, 5), (3, 6)]
```

```
>>> liste = range(10)
>>> print liste
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> map(lambda x: x * 2 + 10, liste)
[10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

Anhand der Beispiele kann man bereits die Syntax und Bedeutung der `lambda`-Funktion erkennen. Wir haben diese noch einmal allgemein dargestellt:



Die `lambda`-Funktion kann auch mit zwei oder mehreren Parametern verwendet werden:



Setze obige drei Beispiele um, ohne die `lambda`-Funktion zu verwenden! Setze unten abgebildetes Beispiel mit der `lambda`-Funktion um!

```
>>> zahlen = [4,21,6,41,42,5,9,7,13,12,60]
>>> def durchDreiTeilbar(x):
        return x % 3 == 0

>>> print filter(durchDreiTeilbar,zahlen)
[21, 6, 42, 9, 12, 60]
```

3.2.7.3 Mastermind

Mit den wenigen Listen-Befehlen, die du bis jetzt gelernt hast, kann man bereits schöne Sachen machen. Sicherlich kennst du das Spiel Mastermind, bei dem man eine vom Spielpartner ausgesuchte Reihenfolge von bunten Kugeln versuchen muss zu erraten.

Spielidee:

Du sollst nun ein ähnliches Spiel entwerfen, nur dass wir nicht mit bunten Kugeln arbeiten, sondern mit den Buchstaben `'A','B','C','D','E'` und `'F'`.
Der Computer soll am Anfang eine Zeichenkette der Länge 4 aus obigen Buchstaben erstellen, etwa `ABCD, AABA, EDDA, ...`. Du sollst nun diese Zeichenkette erraten, indem du selber eine Zeichenkette rätst und der Computer dir sagt, wie viele Buchstaben an der richtigen Stelle sind und wie viele Buchstaben zwar richtig sind, sich aber an der falschen Stelle befinden.

Für diese Aufgabe haben wir zwei verschiedene Aufgabenstellung erstellt, eine mit wenig Vorgaben, die vor allem für Tüftler und Experten gedacht ist, und eine mit vielen Vorgaben, so dass auch Einsteiger die Aufgabe gut bearbeiten können.
Vor dieser Aufgabe sollten auf jeden Fall die Aufgaben zu Listen in Python bearbeitet worden sein.

Aufgabenstellung mit wenig Vorgaben

Zunächst muss sich der `'Computer'` einen geheimen Code erstellen. Dabei kann dir die Methode `random.choice(string)` hilfreich sein, die zufällig einen Buchstaben aus der angegebenen Zeichenkette `string` wählt. Um diese Methode nutzen zu können, musst du das Modul `random` importiert haben.

Nun sollst du solange Raten dürfen, bis du das Codewort erraten hast. Du kannst die Eingabe des Spielers mittels dem Befehl

```
raw_input("Gib deinen Code ein: z.B. ABCD : ").upper()
```

einlesen. Das `.upper()` ist nicht zwingend notwendig, es lässt nur auch Kleinbuchstaben zu, die automatisch in Großbuchstaben umgewandelt werden.
Du musst nun für alle 4 Positionen die Buchstaben des geheimen Codes mit dem geratenen Code vergleichen und ermitteln, wie viele Buchstaben bereits an der richtigen Position sind.
Nun musst du noch ermitteln, wie viele Buchstaben zwar richtig sind, sich aber an der falschen Position befinden. Betrachte hierzu folgendes Beispiel:

Nehmen wir an, dass das Codewort `'AABE'` und das geratene Wort `'BFEA'` ist. Dann erhalten wir die Listen

```
buchstabenInGeraten = [1,1,0,0,1,1]        buchstabenInGeheim = [2,1,0,0,1,0]
```

- Es wurde ein `A` geraten, es sind aber zwei `A`s im Geheimwort. Wie viele Buchstaben wurden richtig geraten?
- Es wurde ein `B` geraten, es ist ein `B` im Geheimwort. Wie viele Buchstaben wurden richtig geraten?
- Es wurde kein `C` geraten, es ist kein `C` im Geheimwort. Wie viele Buchstaben wurden richtig geraten?

- Es wurde kein `D` geraten, es ist kein `D` im Geheimwort. Wie viele Buchstaben wurden richtig geraten?
- Es wurde ein `E` geraten, es ist ein `E` im Geheimwort. Wie viele Buchstaben wurden richtig geraten?
- Es wurde ein `F` geraten, es ist kein `F` im Geheimwort. Wie viele Buchstaben wurden richtig geraten?

Kommst du drauf, wie man anhand der zwei Zahlen für jeden Buchstaben herausfindet, wie viel Buchstaben richtig geraten wurden?

Wenn du weißt, wie viele Buchstaben richtig geraten wurden, musst du von dieser Zahl noch die Anzahl der Buchstaben, die sich bereits an der richtigen Position befinden, abziehen.

Hinweise: Bei der Umsetzung dieser Aufgabe können die Eigenschaften und Methoden von Listen eine große Hilfe sein! Denke an den `map`- und `filter`-Befehl, sowie an die `lambda`-Funktion.

Der Befehl `sum(liste)` summiert alle Einträge einer Liste auf!

Es ist hilfreich, sich eigenen Methoden zu definieren!

Falls du nicht weiterkommst, kannst du dir bei der Aufgabenstellung mit mehr Vorgaben ein paar Tipps holen!

Aufgabenstellung mit mehr Vorgaben

(a) Schreibe eine Methode `erstelleZufallscode()`, die eine Zeichenkette der Länge 4, die nur aus den Buchstaben `'A','B','C','D','E'` und `'F'` besteht, erstellt und zurückgibt.

Hinweise: Der Befehl `random.choice(string)` wählt zufällig einen Buchstaben aus der angegebenen Zeichenkette `string`. Um diesen verwenden zu können, musst du das Modul `random` importieren.

Man kann zwei Zeichenketten mit `+` zusammenfügen:

```
'AB' + 'D' = 'ABD'
```

```
>>> geheimCode = erstelleZufallsCode()
>>> print geheimCode
EAEA
>>> geheimCode = erstelleZufallsCode()
>>> print geheimCode
CEBF
```

(b) Schreibe eine Methode `zaehleBuchstaben(string)`, die das Vorkommen der Buchstaben `'A' - 'F'` in der Zeichenkette `string` zählt und als Liste zurückgibt.
So soll der Methodenaufruf `zaehleBuchstaben('AACF')` die Liste `[2,0,1,0,0,1]` zurückgeben.

Hinweise: `for i in string:` durchläuft nacheinander die Buchstaben der Zeichenkette `string`:

```
>>> string = 'ABC'
>>> for i in string:
        print i


A
B
C
```

Die Methode `string.count(buchstabe)` zählt die Anzahl der Vorkommen des Buchstaben `buchstabe` in der Zeichenkette `string`.

```
>>> liste1 = zaehleBuchstaben('ABCD')
>>> print liste1
[1, 1, 1, 1, 0, 0]
>>> liste2 = zaehleBuchstaben('BBEF')
>>> print liste2
[0, 2, 0, 0, 1, 1]
>>> liste3 = zaehleBuchstaben('CCCC')
>>> print liste3
[0, 0, 4, 0, 0, 0]
```

(c) Wir arbeiten exemplarisch mit den beiden Zeichenketten `geheimCode = 'ABCD'` und `rateCode = 'BBDD'`. Erstelle hieraus eine Liste `vergleichsListe`, deren Elemente Tupel[4] sind, wobei jeweils der erste Buchstabe von `geheimCode` und der erste Buchstabe von `rateCode` ein Tupel bilden, etc.
So soll das Ergebnis wie folgt aussehen:

```
vergleichsListe = [('A','B'),('B','B'),('C','D'),('D','D')]
```

Hinweis: Hierfür ist die `map`-Funktion hilfreich!

(d) Anhand der Variable `vergleichsListe` können wir nun schnell herausfinden, wie viele Buchstaben an die richtige Position geraten wurden. Dazu musst du aus der Liste die Paare herausfiltern, bei denen die erste und zweite Koordinate übereinstimmen. Diese kannst du z. Bsp. in einer Liste `richtigePaare` zwischenspeichern:

```
richtigePaare = [('B','B'),('D','D')]
```

Hinweis: Hierfür ist die `filter`-Funktion hilfreich!

(e) Mit dem Befehl `raw_input("Gib deinen Code ein: z.B. ABCD : ")` wartest du auf eine Benutzereingabe.
Teste diesen Befehl in der IDLE:

```
>>> eingabe = raw_input('Gib deinen Code ein: z.B. ABCD : ')
Gib deinen Code ein: z.B. ABCD : BCDD
>>> print eingabe
BCDD
```

Verwendet man `raw_input("Gib deinen Code ein: z.B. ABCD : ").upper()`, sind auch Kleinbuchstaben zulässig, diese werden automatisch in Großbuchstaben umgewandelt:

```
>>> eingabe = raw_input('Gib deinen Code ein: z.B. ABCD : ').upper()
Gib deinen Code ein: z.B. ABCD : abff
>>> print eingabe
ABFF
```

(f) Wir haben dir bereits die grobe Struktur des Mastermind-Programms vorgegeben. Anhand der Kommentare siehst du, wo du noch Änderungen und Ergänzungen vornehmen musst:

[4] Ein Tupel ist ein Paar von Werten: `(ersterWert,zweiterWert)`

```
1  # Importieren des Random-Moduls fuer die Methode erstelleZufallsCode()
2  import random
3
4  # Methode erstelleZufallsCode, die eine Zeichenkette der Laenge 4 erstellt, die nur aus
       den Buchstaben 'A'-'F' besteht
5  def erstelleZufallsCode():
6      # hier musst du deine Methode aus Aufgabenstellung (a) umsetzen
7
8  # Methode zaehleBuchstaben(string), die die Anzahl der Vorkommen der Buchstaben 'A'-'F'
       zaehlt und in einer Liste zurueckgibt (siehe Teilaufgabe (b))
9  def zaehleBuchstaben(string):
10     # hier musst du deine Methode aus Aufgabenstellung(b) umsetzen
11
12 # Der 'Computer' erzeugt sich einen beliebigen Geheimcode
13 geheimCode = erstelleZufallsCode()
14
15 print("Ich habe mir einen Code aus vier Buchstaben ausgedacht, der nur aus A,B,C,D,E und
       F besteht, z. Bsp. ABCD")
16 print("Buchstaben koennen mehrfach vorkommen")
17 print("Versuche nun, meinen Code zu erraten!")
18
19 # Initialisieren der Variable fuer den geratenen Code - bislang noch kein Code geraten
20 rateCode = ""
21
22 # Wiederholung wird solange ausgefuehrt, wie der Spieler den geheimCode noch nicht
       erraten hat
23 while rateCode != geheimCode:
24     # Einlesen des geratenen Codes - vergleiche Aufgabenstellung (e)
25     rateCode = raw_input("Gib deinen Code ein: z.B. ABCD: ").upper()
26
27     # Stimmt die Anzahl der Zeichen nicht mit 4 ueberein, wird der restliche Teil der
           Wiederholung uebersprungen
28     if len(rateCode) != 4:
29         continue
30
31     # Erstellen der Variable vergleichsListe. Hier musst du die Zeile ergaenzen - vgl.
           Teilaufgabe (c)
32     vergleichsListe = # vergleichsListe erstellen
33
34     # Erstellen der Liste richtigePaare. Hier musst du die Zeile ergaenzen - vgl.
           Teilaufgabe (d)
35     richtigePaare = # richtigePaare erstellen
36
37     # Erstelle hier eine Variable, die die Anzahl der Vorkommen der Buchstaben 'A'-'F'
           im Wort geheimCode in Form einer Liste speichert - das geht dank Vorarbeit ganz
           einfach!
38
39     # Erstelle hier eine Variable, die die Anzahl der Vorkommen der Buchstaben 'A'-'F'
           im Wort rateCode in Form einer Liste speichert - das geht dank Vorarbeit ganz
           einfach!
40
41     # Dem Spieler wird mitgeteilt, wie viele Buchstaben er an der richtigen Position hat
42     print "Richtige Buchstaben an der richten Position : ", # hier musst du die Anzahl
           der Buchstaben an der richtigen Position einfuegen
```

```
43
44      # Dem Spieler wird mitgeteilt, wie viele Buchstaben er zwar richtig, aber an der
            falschen Position hat
45      print "Buchstaben, die zwar richtig sind, aber an der falschen Position: ", # siehe
            Teilaufgabe (g)
46
47  # Hat der Spieler den Geheimcode erraten, wird ihm das mitgeteilt
48  print "Herzlichen Glueckwunsch! Du hast den richtigen Code erraten: ", geheimCode
```

(g) Das Mastermind-Programm ist fast fertig. Du musst dem Spieler noch mitteilen, wie viele Buchstaben er zwar richtig hat, aber an der falschen Position.
Das kann man mit einem kleinen Trick ganz schnell herausfinden - du benötigst dazu die zwei Listen, in denen steht, wie oft jeder Buchstabe in der Zeichenkette vorkommt. Diese hast du in Zeile 38 und 43 erzeugt.

Nehmen wir an, dass das Codewort `'AABE'` und das geratene Wort `'BFEA'` ist. Dann erhalten wir die Listen

`buchstabenInGeraten = [1,1,0,0,1,1]` `buchstabenInGeheim = [2,1,0,0,1,0]`

- Es wurde ein `A` geraten, es sind aber zwei `A`s im Geheimwort. Wie viele Buchstaben wurden richtig geraten?
- Es wurde ein `B` geraten, es ist ein `B` im Geheimwort. Wie viele Buchstaben wurden richtig geraten?
- Es wurde kein `C` geraten, es ist kein `C` im Geheimwort. Wie viele Buchstaben wurden richtig geraten?
- Es wurde kein `D` geraten, es ist kein `D` im Geheimwort. Wie viele Buchstaben wurden richtig geraten?
- Es wurde ein `E` geraten, es ist ein `E` im Geheimwort. Wie viele Buchstaben wurden richtig geraten?
- Es wurde ein `F` geraten, es ist kein `F` im Geheimwort. Wie viele Buchstaben wurden richtig geraten?

Kommst du drauf, wie man anhand der zwei Zahlen für jeden Buchstaben herausfindet, wie viel Buchstaben richtig geraten wurden?

Man nimmt immer das Minimum der beiden Zahlen für jeden Buchstaben!

```
richtigeBuchstaben = min(1,2) + min(1,1) + min(0,0) + min(0,0) + min(1,1) + min(1,0)
```

Wenn du nun von dieser Zahl noch die Anzahl der Buchstaben abziehst, die an der richtigen Position sind, hast du die Anzahl der Buchstaben, die richtig sind, aber an der falschen Position.

Du hast bereits die beiden Listen mit den Zahlen erzeugt und weißt jetzt, wie du daraus die gesuchte Anzahl berechnest. Setze dies nun in Python um und ergänze dein Programm entsprechend!

Hinweise: Du kannst hierfür entweder eine neue Methode schreiben oder wieder mit der `map`-Funktion arbeiten. Schaffst du auch beide Varianten?

Der Befehl `sum(liste)` summiert alle Einträge einer Liste auf!

Dein Programm ist nun fertig!

Unten ist noch ein möglicher Ablauf des Mastermind-Spiels dargestellt:

```
>>>
Ich habe mir einen Code aus vier Buchstaben ausgedacht, der nur aus A,B,C,D,E und F besteht, z.Bsp. ABCD
Buchstaben koennen mehrfach vorkommen!
Versuche nun, meinen Code zu erraten!
Gib deinen Code ein: z.B. ABCD : AAAA
Richtige Buchstaben an der richtigen Position :  1
Buchstaben, die zwar richtig sind, aber an der falschen Position :  0
##############################################################################################
Gib deinen Code ein: z.B. ABCD : ABBB
Richtige Buchstaben an der richtigen Position :  0
Buchstaben, die zwar richtig sind, aber an der falschen Position :  1
##############################################################################################
Gib deinen Code ein: z.B. ABCD : CCAA
Richtige Buchstaben an der richtigen Position :  1
Buchstaben, die zwar richtig sind, aber an der falschen Position :  1
##############################################################################################
Gib deinen Code ein: z.B. ABCD : DDAC
Richtige Buchstaben an der richtigen Position :  0
Buchstaben, die zwar richtig sind, aber an der falschen Position :  3
##############################################################################################
Gib deinen Code ein: z.B. ABCD : CADE
Richtige Buchstaben an der richtigen Position :  4
Buchstaben, die zwar richtig sind, aber an der falschen Position :  0
##############################################################################################
Herzlichen Glueckwunsch! Du hast den richtigen Code erraten :  CADE
```

Abbildung 3.4: Möglicher Ablauf des Mastermind-Spiels

Viel Spaß beim Spielen!

3.3 Lösungen

In diesem Abschnitt finden Sie Lösungsvorschläge zu den Aufgaben aus Abschnitt 3.2.

3.3.1 Sequenz und Wiederholung

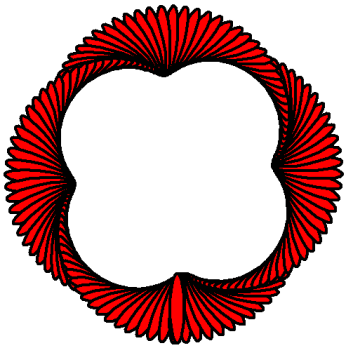
3.3.1.1 Wiederholung mit fester Anzahl

(a) `Quadrat.py`:

```
1 import turtle
2
3 turtle.shape("turtle")
4
5 for i in range(4):
6     turtle.forward(100)
7     turtle.left(90)
```

(b) **Lösungsbeispiel 1**:

```
1  import turtle
2
3  turtle.shape("circle")
4  turtle.fillcolor("red")
5  turtle.shapesize(5,1,4)
6
7  for i in range(120):
8      turtle.left(3)
9      turtle.forward(10)
10     turtle.stamp()
11     turtle.tilt(6)
```



Lösungsbeispiel 2:

```
1  import turtle
2
3  turtle.shape("circle")
4  turtle.fillcolor("green")
5  turtle.shapesize(5,1,4)
6
7  for i in range(120):
8      turtle.left(8)
9      turtle.forward(3)
10     turtle.right(7)
11     turtle.stamp()
12     turtle.right(4)
13     turtle.tilt(12)
```

3.3.2 Eigene Methoden definieren

3.3.2.1 Bilderrahmen

`Bilderrahmen.py`:

```
import turtle

turtle.shape("circle")

# Definition der Methode bilderrahmen()
def bilderrahmen():
    turtle.pensize(10)

    for i in range(2):
    turtle.forward(200)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)

# Aufruf der Methode bilderrahmen()
bilderrahmen()
```

3.3.2.2 Mehrere Kreise umeinander

`Kreise.py`:

```
import turtle

turtle.shape("circle")

# Definition der Methode kreise()
def kreise():
    for i in range(5):
        turtle.circle(15*(i+1))
        turtle.penup()
        turtle.right(90)
        turtle.forward(15)
        turtle.left(90)
        turtle.pendown()

# Aufruf der Methode kreise()
kreise()
```

3.3.2.3 n-Eck

`nEck.py`:

```
import turtle

turtle.shape("circle")

# Definition der Methoden zeichne_neck(n)
```

```
6  def zeichne_neck(n):
7      for i in range(n):
8          turtle.forward(100)
9          turtle.left(360/n)
10
11 # Testen der Methode zeichne_neck(n)
12 zeichne_neck(3)
13 zeichne_neck(4)
14 zeichne_neck(8)
```

3.3.3 Methoden (bzw. Funktionen) können als Argumente von Methoden (bzw. Funktionen) vorkommen

3.3.3.1 Drucken

Lösungsbeispiel:

```
1  import random
2
3  # Funktion ohne Parameter
4  def getRandomWert():
5      return random.random()
6
7  # Funktion mit zwei Parameter
8  def getRandomWertInGrenzen(x,y):
9      zufall = random.random()*y
10     while zufall < x:
11         zufall = random.random()*y
12     return zufall
13
14 def drucken0(f):
15     print("Ausgabe des Rueckgabewertes einer Funktion ohne Parameter: " + str(f()))
16
17 def drucken2(f,x,y):
18     print("Zufallswert zwischen den Grenzen " + str(x) + " und " + str(y) + ": " +
           str(f(x,y)))
```

3.3.4 Funktionen können als Werte von Funktionen vorkommen

3.3.4.1 Ableitungen

(a) `ableitung_quadrat.py`:

```
1  # f muss eine Funktion mit einem formalen Parameter sein
2  def ableitung_von(f):
3      dx = 1.e-6
4      def d_nach_dx_von_Argument(x):
5          return (f(x+dx)-f(x))/dx
6      return d_nach_dx_von_Argument
7
8  # Quadrat-Funktion
9  def quadrat(x):
10     return x*x
```

```
11
12 # Testen der selbstdefinierten Methoden
13 print(ableitung_von(quadrat)(0))
14 print(ableitung_von(quadrat)(1))
15 print(ableitung_von(quadrat)(2))
16 print(ableitung_von(quadrat)(4))
```

(b) `zweite_ableitung.py`:

```
1  # Importieren der trigonometrischen Funktion Sinus und Kosinus
2  from math import sin,cos
3
4  # f muss eine Funktion mit einem formalen Parameter sein
5  def ableitung_von(f):
6      dx = 1.e-6
7      def d_nach_dx_von_Argument(x):
8          return (f(x+dx)-f(x))/dx
9      return d_nach_dx_von_Argument
10
11 # f muss eine Funktion mit einem formalen Parameter sein
12 def zweite_ableitung_von(f):
13     return ableitung_von(ableitung_von(f))
14
15 # Quadrat-Funktion
16 def quadrat(x):
17     return x*x
18
19 # Testen der selbstdefinierten Methoden
20 print(ableitung_von(quadrat)(0))
21 print(zweite_ableitung_von(quadrat)(0))
22 print(zweite_ableitung_von(quadrat)(1))
23 print(zweite_ableitung_von(quadrat)(2))
24 print(zweite_ableitung_von(sin)(0))
25 print(zweite_ableitung_von(cos)(0))
```

(c) `funktionen_zeichnen.py`:

```
1  import turtle
2  from turtle import Screen
3  from math import sin,cos
4
5  # Zeichnen des Graphen der Funktion f
6  def graph(f):
7      screen = Screen()
8      screen.setworldcoordinates(-4,-2,4,2)
9      x = -4
10     turtle.penup()
11     turtle.goto(x,f(x))
12     turtle.pendown()
13
14     while x < 4:
```

```
15          x = x+0.1
16          turtle.goto(x,f(x))
17
18  # Quadrat-Funktion
19  def quadrat(x):
20      return x*x
21
22  # Zeichnen verschiedener Funktionen
23  graph(quadrat)
24  graph(sin)
25  graph(cos)
```

(d) `abl_zeichnen.py`:

```
1   import turtle
2   from turtle import Screen
3   from math import sin,cos
4
5   # f muss eine Funktion mit einem formalen Parameter sein
6   def ableitung_von(f):
7       dx = 1.e-6
8       def d_nach_dx_von_Argument(x):
9           return (f(x+dx)-f(x))/dx
10      return d_nach_dx_von_Argument
11
12  # Zeichnen des Graphen der Funktion f
13  def ablgraph(f):
14      screen = Screen()
15      screen.setworldcoordinates(-4,-2,4,2)
16      x = -4
17      turtle.penup()
18      turtle.goto(x,ableitung_von(f)(x))
19      turtle.pendown()
20
21      while x < 4:
22          x = x+0.1
23          turtle.goto(x,ableitung_von(f)(x))
24
25  # Quadrat-Funktion
26  def quadrat(x):
27      return x*x
28
29  # Zeichnen verschiedener Funktionen
30  graph(quadrat)
31  graph(sin)
32  graph(cos)
```

3.3.5 Ereignisse und bedingte Anweisung

3.3.5.1 Schildkröte mit der Maustaste drehen

Man muss folgende vier Fälle unterscheiden:

Hieraus resultiert folgendes Programm:

```
import turtle

def drehen(x,y):
    if turtle.heading() == 0:
        if y >=0:
            turtle.left(90)
        else:
            turtle.right(90)
    elif turtle.heading() == 90:
        if x >=0:
            turtle.right(90)
        else:
            turtle.left(90)
    elif turtle.heading() == 180:
        if y >=0:
            turtle.right(90)
        else:
            turtle.left(90)
    elif turtle.heading() == 270:
        if x >= 0:
            turtle.left(90)
        else:
            turtle.right(90)

turtle.shape("turtle")
turtle.shapesize(3,3,3)
turtle.onclick(drehen)

turtle.mainloop()
```

3.3.5.2 Schildkröte mit zwei Farbzuständen

`farbwechsel.py`:

```
import turtle

def farbewechseln(x,y):
    if turtle.fillcolor() == "black":
        turtle.color("red")
```

```
6       else:
7           turtle.color("black")
8
9   turtle.shape("turtle")
10  turtle.shapesize(3,3,3)
11  turtle.onclick(farbewechseln)
12
13  turtle.mainloop()
```

Beachte: Da wir die Methode `farbewechseln` der `onclick`-Methode übergeben wollen, müssen wir diese mit den zwei Parameter `x` und `y` definieren, auch wenn wir die Parameter nicht brauchen. Lässt man die Parameter `x` und `y` weg, bekommen wir eine Fehlermeldung, da die `onclick`-Methode eine Methode mit zwei Parametern erwartet!

3.3.5.3 Schildkröte folgt dem Mauszeiger und zeichnet

`zeichnen.py`:

```
1   import turtle
2   from turtle import Screen
3
4   # Stift anheben, falls abgesenkt und Stift absenken, falls angehoben
5   # Wichtig sind wieder die beiden Parameter x und y fuer die onclick-Methode
6   def zeichnen(x,y):
7       if turtle.isdown():
8           turtle.penup()
9       else:
10          turtle.pendown()
11
12  turtle.shape("turtle")
13  turtle.shapesize(3,3,3)
14  screen = Screen()
15  # Der Parameter 1 detektiert einen Linksklick
16  screen.onclick(turtle.goto,1)
17  # Der Parameter 3 detektiert einen Rechtsklick
18  turtle.onclick(zeichnen,3)
19
20  turtle.mainloop()
```

3.3.6 Erstellen einer eigenen Klasse

3.3.6.1 Die Klasse Vieleck

Die Methoden `zeichenWinkel(self)` und `zeichnen(self)`

```
1 def zeichenWinkel(self):
2     return (360/self.ecken)
3
4 def zeichnen(self):
5     for i in range(self.ecken):
6         self.turtle.forward(self.laenge)
7         self.turtle.left(self.zeichenWinkel())
```

Mittelpunkt einzeichnen

Wir definieren uns zunächst eine Hilfsmethode, die die Höhe des gleichschenkligen Dreiecks berechnet. Zunächst bestimmen wir den Wert des Basiswinkels α im gleichschenkligen Dreieck. Es gilt:

$$\alpha = \frac{180° - \texttt{zeichenWinkel()}}{2}$$

Mit Hilfe von α können wir nun die Höhe berechnen:

$$\begin{aligned} \tan\alpha &= \frac{h}{\frac{s}{2}} \\ \Longrightarrow \qquad h &= \frac{s}{2}\cdot\tan\alpha \\ h &= \frac{s}{2}\cdot\tan\left(\frac{180° - \texttt{zeichenWinkel()}}{2}\right) \end{aligned}$$

Wir müssen nur noch berücksichtigen, dass die trigonometrischen Funktionen in Python als Argument einen Winkel im Bogenmaß erwarten:

```
1 def berechneHoehe(self):
2     return (self.laenge/2)*(math.tan((180-self.zeichenWinkel())/2*math.pi/180))
```

Mit der Hilfsmethode `berechneHoehe()` können wir nun auch die Methode `zeichneMittelpunkt(self)` implementieren:

```
1  def zeichneMittelpunkt(self):
2      self.turtle.penup()
3      # Turtle geht zum Mittelpunkt der Seite
4      self.turtle.forward(self.laenge/2)
5      self.turtle.left(90)
6      # Turtle geht um den Wert der Hoehe senkrecht zur Seite zum Mittelpunkt
7      self.turtle.forward(self.berechneHoehe())
8      self.turtle.right(90)
9      self.turtle.pendown()
10     # Mittelpunkt wird mit einem kleinen Kreis markiert
11     self.turtle.circle(2)
```

Stern zeichnen

Zunächst berechnen den Drehwinkel bei (1): Wir kennen bereits den Basiswinkel α im gleichschenkligen Dreieck. Aus Symmetriegründen ist dies unser gesuchter Drehwinkel:

```
1 def berechneDrehwinkel1(self):
2     return (180-self.zeichenWinkel())/2
```

Als nächstes Berechnen wir die Seitenlänge x bei (2): Diese kann man mit dem Satz des Pythagoras aus der Höhe h und der halben Seitenlänge $\frac{s}{2}$ des regelmäßigen Vielecks bestimmen. Es gilt:

$$\begin{aligned} x^2 &= h^2 + \left(\frac{s}{2}\right)^2 \\ x &= \sqrt{h^2 + \left(\frac{s}{2}\right)^2} \end{aligned}$$

Folglich erhalten wir:

```
1 def berechneSeitenlaenge2(self):
2     return math.sqrt(self.berechneHoehe()**2+(self.laenge/2)**2)
```

Aus unseren bisherigen Überlegungen wissen wir, dass der Drehwinkel bei (3) gerade der Zeichenwinkel des regelmäßigen Vielecks ist. Es gilt also:

```
1 def berechneDrehwinkel3(self):
2     return 180-(180-2*self.berechneDrehwinkel1())
```

Die Seitenlänge bei (4) ist analog zu (2) - Eigenschaften eines gleichschenkligen Dreiecks. Mir müssen also nur noch den Drehwinkel bei (5) berechnen. Dies ist gerade die Differenz `zeichenWinkel() - berechneDrehWinkel1()`:

```
1 def berechneDrehwinkel5(self):
2     return self.zeichenWinkel()-self.berechneDrehwinkel1()
```

Die Implementation der Methode `zeichneStern()` gelingt nun mit Verwendung der vier Hilfsmethoden:

```
1 def zeichneStern(self):
2     for i in range(self.ecken):
3         self.turtle.color("blue")
4         self.turtle.right(self.berechneDrehwinkel1())
5         self.turtle.forward(self.berechneSeitenlaenge2())
6         self.turtle.left(self.berechneDrehwinkel3())
7         self.turtle.forward(self.berechneSeitenlaenge2())
8         self.turtle.left(self.berechneDrehwinkel5())
```

3.3.7 Listen in Python

3.3.7.1 Listen verschmelzen

```
1 def verschmelzeListen(liste1,liste2):
2     while len(liste2) > 0:
3         liste1.append(liste2.pop())
```

Setzt man die Methode `verschmelzeListen(liste1,liste2)` wie oben um und ruft diese mit den Listen `liste1 = [1,2,3]` und `liste2 = [4,5,6]` auf, erhält man folgende Listen: `liste1 = [1,2,3,4,5,6]` und `liste2 = []`.

Möchte man die zwei Listen, die als Parameter übergeben werden, unverändert lassen, könnte man die Methode wie folgt umsetzen:

```
1  import copy
2
3  def verschmelzeListen(liste1,liste2):
4      hlist1 = copy.copy(liste1)
5      hlist2 = copy.copy(liste2)
6
7      while len(hliste2) > 0:
8          hliste1.append(hliste2.pop())
9
10     return hlist1
```

Diese Umsetzung liefert folgendes Ergebnis:

```
>>> list1 = [1,2,3]
>>> list2 = [4,5]
>>> list3 = verschmelzeListen(list1,list2)
>>> list1
[1, 2, 3]
>>> list2
[4, 5]
>>> list3
[1, 2, 3, 4, 5]
```

3.3.7.2 Die filter- und map-Funktion

```
1 def durchDreiTeilbar(zahl):
2     return zahl % 3 == 0
```

Anwenden der `filter`-Funktion auf die Liste der Zahlen von 1 bis 30 liefert:

```
>>> filter(durchDreiTeilbar, range(1,30))
[3, 6, 9, 12, 15, 18, 21, 24, 27]
```

Kubik-Zahlen von 1 bis 15 als Liste:

```
1 def kubik(x):
2     return x*x*x
3
4 map(kubik,range(1,15))
```

Listen ohne `lambda`-Funktion erstellen:

(1)

```
1 def kubik(x):
2     return x*x*x
3
4 map(kubik,range(10))
```

(2)

```
1 def tupel(x,y):
2     return (x,y)
3
4 map(tupel,[1,2,3],[4,5,6])
```

(3)

```
1 def transform(x):
2     return x*2+10
3
4 map(transform,range(10))
```

Liste mit `lambda`-Funktion erstellen:

```
1 zahlen = [4,31,6,41,42,5,9,7,13,12,60]
2 filter(lambda x: x % 3 == 0, zahlen)
```

3.3.7.3 Mastermind

Wir stellen hier die Lösung der Aufgabenstellung mit mehr Vorgaben vor:

(a) Methode `erstelleZufallsCode()`:

```
1 def erstelleZufallsCode():
2     code = ''
3     for i in range(4):
4         code = code + random.choice('ABCDEF')
5     return code
```

(b) Methode `zaehleBuchstaben(string)`:

```
1 def zaehleBuchstaben(string):
2     vorkommen = []
3     for i in 'ABCDEF':
4         vorkommen.append(string.count(i))
5     return vorkommen
```

(c) Erstellen der Liste `vergleichsListe`:

```
1 vergleichsListe = map(lambda x,y: (x,y), geheimCode,rateCode)
```

(d) Erstellen der Liste `richtigePaare`:

```
richtigePaare = filter(lambda (x,y): x==y, vergleichsListe)
```

(e) Ausprobieren!

(f) Das fertige Programm:

```
# Importieren des Random-Moduls fuer die Methode erstelleZufallsCode()
import random

# Methode erstelleZufallsCode(), die eine Zeichenkette der Laenge 4 erstellt, die nur aus den Buchstaben 'A'-'F' besteht
def erstelleZufallsCode():
    code = ""
    while len(code) < 4:
        code = code + random.choice("ABCDEF")
    return code

# Methode zaehleBuchstaben(string), die die Anzahl der Vorkommen der Buchstaben 'A'-'F' zaehlt und in einer Liste zurueckgibt
def zaehleBuchstaben(string):
    liste = []
    for i in 'ABCDEF':
        liste.append(string.count(i))
    return liste

# Der 'Computer' erzeugt sich einen beliebigen Geheimcode
geheimCode = erstelleZufallsCode()

print "Ich habe mir einen Code aus vier Buchstaben ausgedacht, der nur aus A,B,C,D,E und F besteht, z.Bsp. ABCD"
print "Buchstaben koennen mehrfach vorkommen!"
print "Versuche nun, meinen Code zu erraten!"

# Initialisieren der Variable fuer den geratenen Code - bislang noch kein Code geraten
rateCode = ""

# Wiederholung wird solange ausgefuehrt, wie der Spieler den geheimCode noch nicht erraten hat
while rateCode != geheimCode:
    # Einlesen des geratenen Codes
    rateCode = raw_input("Gib deinen Code ein: z.B. ABCD : ").upper()

    # Stimmt die Anzahl der Zeichen nicht mit 4 ueberein, wird der restliche Teil der Wiederholung uebersprungen
    if len(rateCode) != 4:
        continue

    # Erstellen der Liste vergleichsliste
    vergleichsListe = map(lambda x,y : (x,y), rateCode,geheimCode)

    # Erstellen der Liste richtigePaare
```

```
41      richtigePaare = filter(lambda (x,y) : x==y, vergleichsListe)
42
43      # Erstellen der Liste buchstabenInGeraten, die die Anzahl der einzelnen Buchstaben
            in der Zeichenkette rateCode in Form einer Liste speichert
44      buchstabenInGeraten = zaehleBuchstaben(rateCode)
45
46      # Erstellen der Liste buchstabenInGeheim, die die Anzahl der einzelnen Buchstaben in
            der Zeichenkette geheimCode in Form einer Liste speichert
47      buchstabenInGeheim = zaehleBuchstaben(geheimCode)
48
49      # Erstellen einer Liste gleicheBuchstaben, die enthaelt, welche Buchstaben richtig
            geraten wurden
50      gleicheBuchstaben = map(lambda x,y : min(x,y), buchstabenInGeraten,
            buchstabenInGeheim)
51
52      # Dem Spieler wird mitgeteilt, wie viele Buchstaben er an der richtigen Position hat
53      print "Richtige Buchstaben an der richtigen Position : ", len(richtigePaare)
54
55      # Dem Spieler wird mitgeteilt, wie viele Buchstaben er zwar richtig, aber an der
            falschen Position hat
56      print "Buchstaben, die zwar richtig sind, aber an der falschen Position : ",
            sum(gleicheBuchstaben) - len(richtigePaare)
57      print "############################################################"
58
59  # Hat der Spieler den Geheimcode erraten, wird ihm das mitgeteilt
60  print "Herzlichen Glueckwunsch! Du hast den richtigen Code erraten : ", geheimCode
```

(g) Wie in der Aufgabenstellung beschrieben verwenden wir die beiden Listen `buchstabenInGeraten` und `buchstabenInGeheim`. Mit diesen beiden Listen kann man die Anzahl der Buchstaben, die zwar richtig, aber an der falschen Position sind, mit Hilfe des Minimums der beiden Zahlen für jeden Buchstaben berechnen. Die Minima müssen wir aufsummieren und davon noch die Anzahl der Buchstaben abziehen, die an der richtigen Position sind:

```
1  # Minimum fuer alle Buchstabenpaare berechnen
2  gleicheBuchstaben = map(lambda x,y: min(x,y), buchstabenInGeraten, buchstabenInGeheim)
3
4  # Die Minima werden aufsummiert und davon die Anzahl der richtig positionierten
       Buchstaben abgezogen
5  print "Buchstaben, die zwar richtig sind, aber an der falschen Position : ",
       sum(gleicheBuchstaben) - len(richtigePaare)
```

Kapitel 4

Physical Computing

Überblick:

4.1 Adafruit-Library und Editor

In Aufgabe 4.3.1 werden wir den Raspberry Pi nur als „eine Art Batterie“ mit zwei Anschlüssen (3,3 Volt und Erde) verwenden. Ab Aufgabe 4.3.2 werden wir die Programmiersprache Python verwenden, um an einzelnen Pins Spannung an- oder abschalten zu können.
Dazu brauchen wir Software, die uns die GPIO Pins via Python steuern lässt.

Falls diese Software noch nicht installiert wurde, müssen Sie folgende 5 Schritte durchführen:

(1) Installation von GIT:

```
1  pi@raspberrypi ~ $ sudo apt-get install git
```

(2) In den gewünschten Ordner navigieren - in diesen wird installiert. Der Ordner kann beliebig gewählt werden, man muss ihn später nur immer wieder finden können

(3) Source Files beschaffen:

```
1  pi@raspberrypi ~ $ git clone http://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git
```

(4) Python Development Kit installieren:

```
1  pi@raspberrypi ~ $ sudo apt-get install python-dev
```

(5) Rpi.GPIO installieren:

```
1  pi@raspberrypi ~ $ sudo apt-get install python-rpi.GPIO
```

Nach der Installation erstellen wir ein neues Python Script. Als Editor kann GEANY statt IDLE verwendet werden, da GEANY etwas komfortabler zu bedienen ist. Wurde GEANY noch nicht installiert, müssen Sie dies noch schnell erledigen:

```
1  pi@raspberrypi ~ $ sudo apt-get install geany
```

Der Editor GEANY muss als `root` gestartet werden, da Python sonst keinen Zugriff auf die GPIO Pins des Raspberry Pi bekommt. Dies erreichen Sie mit folgendem Befehl:

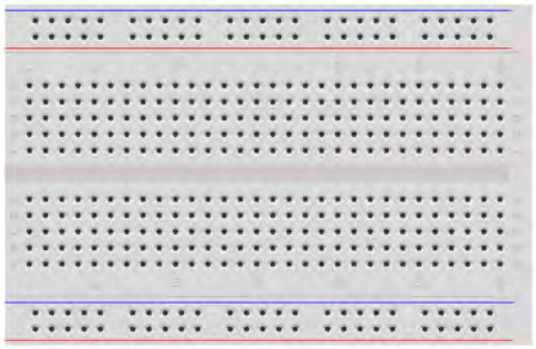
```
1  pi@raspberrypi ~ $ sudo geany
```

4.2 Breadboard, Cobbler und Schaltelemente

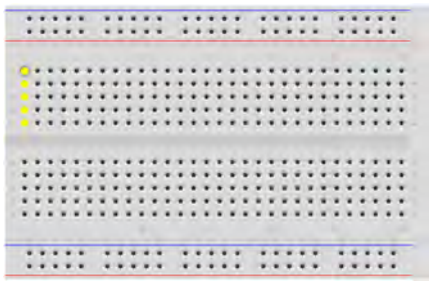
In diesem Abschnitt gehen wir kurz auf die Funktionsweise und Eigenschaften des Breadboards, Cobblers und der einzelnen Schaltelemente ein. In Schülerlaborkursen an der Universität Passau ist uns aufgefallen, dass Schülerinnen und Schülern bei der Bearbeitung der Aufgaben große Probleme hatten. Dies lag nicht nur daran, dass sie aufgrund der Verwendung von Python mit einer ihnen großteils unbekannten Programmiersprache konfrontiert wurden, sondern auch am Verständnis und Umgang mit den Breadboards und Schaltelementen.
Um die Einstiegshürde „abzuflachen“ entwickelten wir einen Foliensatz zum grundlegenden Einstieg in die Thematik, bei dem, neben Grundgedanken, auch die einzelnen Schaltelemente und ihre Besonderheiten aufgegriffen werden. Zudem beinhaltet jener viele Beispielschaltungen, bei denen die Schülerinnen und Schüler ihr Verständnis überprüfen können. Die Kernaussagen dieses Foliensatzes haben wir in diesem Abschnitt zusammengefasst:

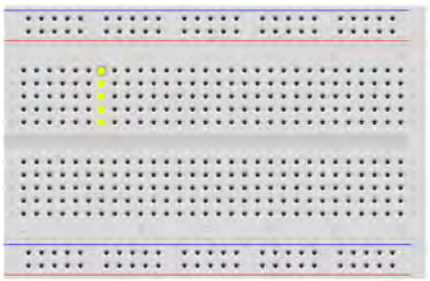
4.2.1 Breadboard

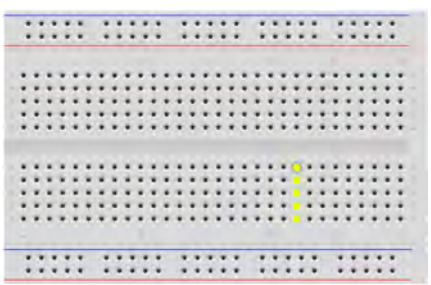
Um die Schaltpläne in einer echten Schaltung umsetzen zu können, verwenden wir ein `Streckbrett (Breadboard)`:

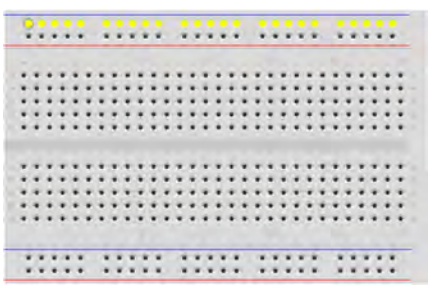


Auf dem Breadboard sind bestimmte Spalten und Zeilen **leitend miteinander verbunden**! Exemplarisch dazu folgende vier Bilder:



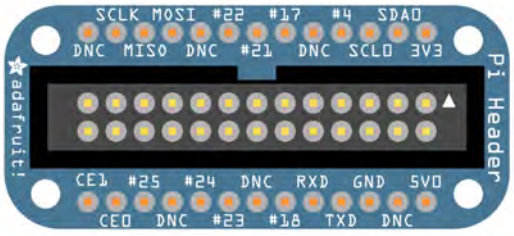




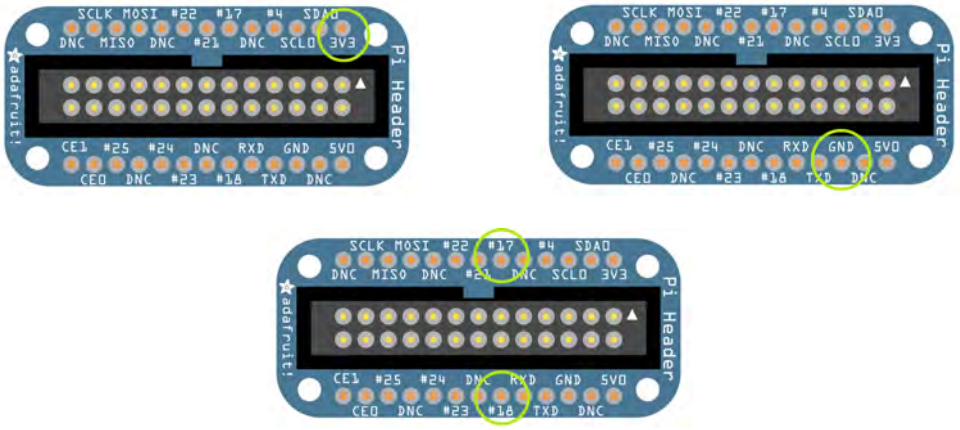


4.2.2 Cobbler

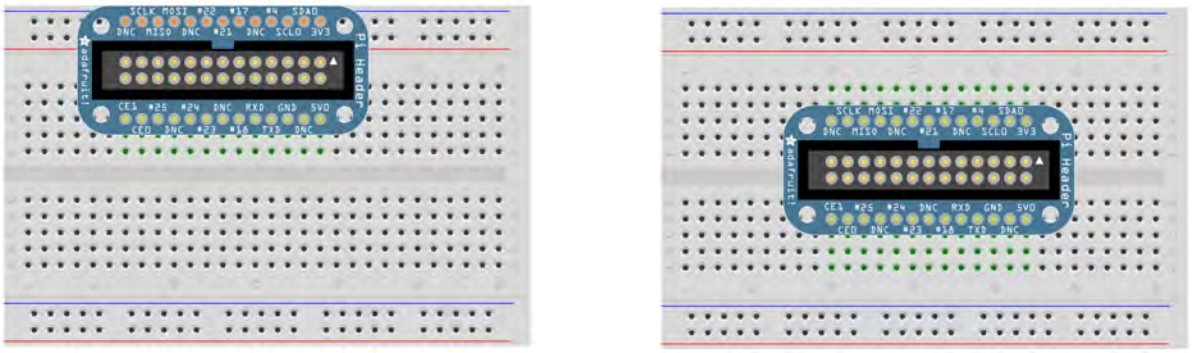
Um die GPIO Pins des Raspberry Pi für eigene Schaltungen besser nutzen zu können, sind diese über einen `Cobbler` mit dem Breadboard verbunden:



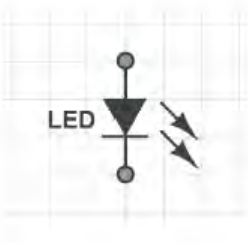
Der Raspberry Pi bietet einen Pin `3V3` mit ständiger Stromquelle mit konstanter Spannung 3,3 V (links oben), einen Pin `Ground` (`GND`), was im Deutschen soviel wie Erde (Masse) bedeutet (oben rechts) und zusätzlich jede Menge weitere Pins, wie etwa 17 und 18. An diesen kann man z. Bsp. via Python-Programm gezielt Spannung anlegen oder Spannungsabfall messen (unten Mitte):



Es ist wichtig, den Cobbler richtig mit dem Breadboard zu verbinden; ansonsten kann es zu einem Kurzschluss kommen. Beim Verbinden des Cobblers mit dem Breadboard muss man darauf achten, dass die einzelnen Pins des Cobblers nicht leitend verbunden sind! Erinnert man sich an die Konfiguration des Breadboards, erkennt man, dass der Cobbler auf dem linken Bild **falsch** und auf dem rechten Bild **richtig** mit dem Breadboard verbunden wurde:



4.2.3 Leuchtdiode (LED)

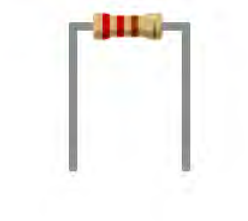


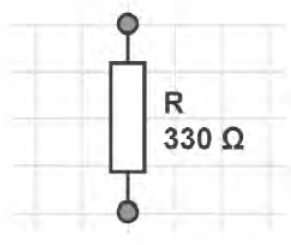
- es macht einen Unterschied, wie herum die LED in die Schaltung eingesetzt wird!

  Dioden lassen den Strom nur in eine Richtung durch

- die Minus-Seite (Kathode) erkennt man an dem kürzeren Fuß
- die Minus-Seite muss in Richtung `Ground` (`GND`) zeigen
- eine LED ist immer **in Reihe** mit einem **Schutzwiderstand** zu schalten, der den Strom begrenzt!
- fließt ein zu großer Strom durch die LED, so „brennt diese durch“!

4.2.4 Widerstand

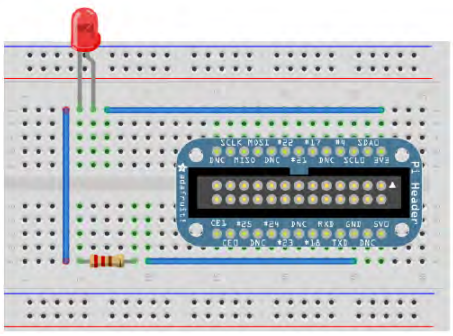


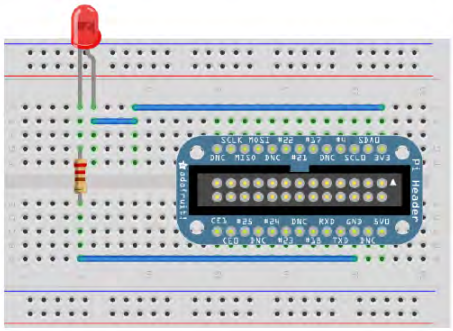


- verbindet man eine Spannungsquelle mit `GND` ohne Widerstand, gibt es einen **Kurzschluss** und der Raspberry Pi stürzt ab
- in unseren Schaltungen verwenden wir nur 330 Ohm und 3,3 kOhm Widerstände
- vor dem **Schließen des Stromkreises** auf jeden Fall noch mal darauf achten, ob sich ein **Widerstand in der Schaltung** befindet!!
- Achtet beim Stecken von Widerständen auf das Breadboard und welche Pins leitend verbunden sind!

4.2.5 Beispielschaltungen und Lösungen

Um überprüfen zu können, ob die Schülerinnen und Schüler die Kernaussagen der bisherigen Abschnitte verstanden haben, folgen im Foliensatz acht Beispielschaltungen. Unter diesen Beispielschaltungen befinden sich fehlerhafte und korrekte Schaltungen. Die Schülerinnen und Schüler sollen nun anhand des bereits erworbenen Wissens begründen, warum eine Schaltung fehlerhaft / korrekt ist.
Zudem sollen die Schülerinnen und Schüler angeben, ob die LED bei den jeweiligen Schaltungen durchbrennt und ob der Raspberry Pi kurzgeschlossen wird.



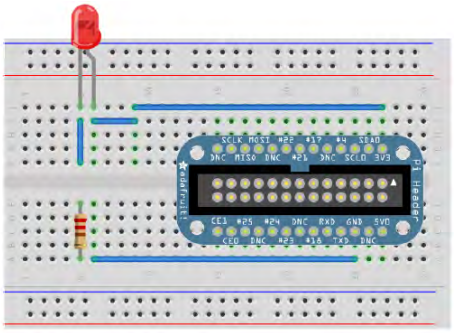


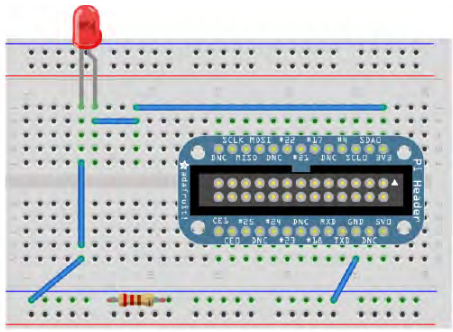
Linke Schaltung:

- kein geschlossener Stromkreis, da nicht einmal die erste Leitung mit der LED leitend verbunden ist
- LED brennt folglich nicht durch
- Raspberry Pi wird nicht kurz geschlossen

Rechte Schaltung:

- geschlossener Stromkreis
- LED brennt nicht durch, da sich ein Schutzwiderstand in der Schaltung befindet
- Raspberry Pi wird nicht kurzgeschlossen



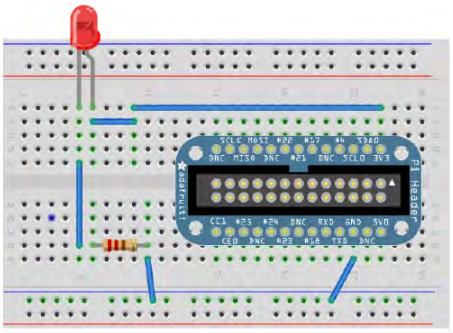


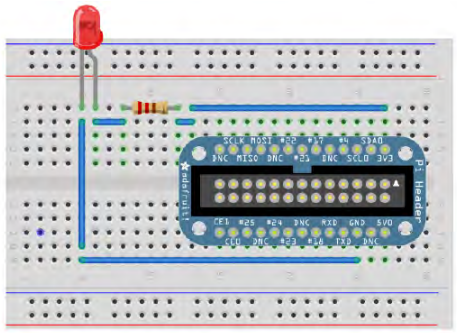
Linke Schaltung:

- kein geschlossener Stromkreis, da z. Bsp. die LED nicht leitend mit dem Widerstand verbunden ist
- LED brennt folglich nicht durch
- Raspberry Pi wird nicht kurz geschlossen

Rechte Schaltung:

- geschlossener Stromkreis
- LED brennt durch; der Schutzwiderstand hat keinen Effekt, da alle Pins in der unteren Zeile leitend miteinander verbunden sind!
- Raspberry Pi wird kurzgeschlossen



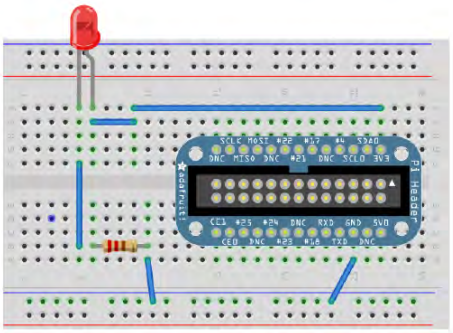


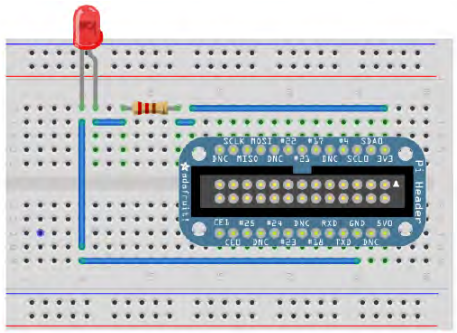
Linke Schaltung:

- geschlossener Stromkreis!
- LED brennt nicht durch, da sich ein Schutzwiderstand in der Schaltung befindet
- Raspberry Pi wird nicht kurzgeschlossen

Rechte Schaltung:

- geschlossener Stromkreis
- LED brennt nicht durch, da sich ein Schutzwiderstand in der Schaltung befindet - kein Unterschied ob sich dieser vor oder nach der LED in der Schaltung befindet
- Raspberry Pi wird nicht kurzgeschlossen





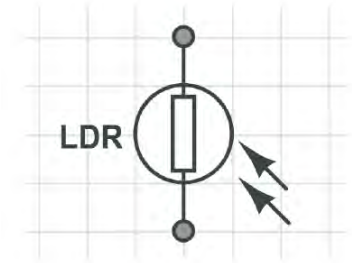
Linke Schaltung:

- geschlossener Stromkreis
- LED brennt durch; der Schutzwiderstand hat keinen Effekt, da alle Pins in der Spalte leitend miteinander verbunden sind!
- Raspberry Pi wird kurzgeschlossen

Rechte Schaltung:

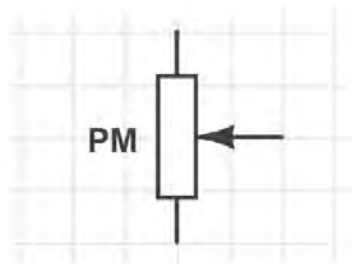
- geschlossener Stromkreis
- LED brennt nicht durch, da sich ein Schutzwiderstand in der Schaltung befindet
- Raspberry Pi wird nicht kurzgeschlossen

4.2.6 Lichtwiderstand (LDR)



- ist ein **lichtabhängiger Widerstand**
- besteht aus einem Halbleiter
- je höher der Lichteinfall, umso kleiner wird der elektrische Widerstand (**innerer fotoelektrischer Effekt**)
- je niedriger der Lichteinfall, umso größer wird der elektrische Widerstand
- auch bei starkem Licht hat der LDR noch genügend Widerstand, um als alleiniger Widerstand einen Kurzschluss zu vermeiden
- Achtet beim Stecken von Widerständen und LDRs auf das Breadboard und welche Pins leitend verbunden sind!

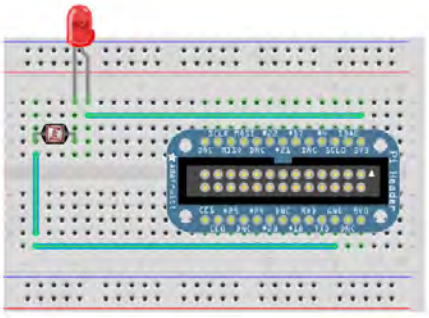
4.2.7 Potentiometer

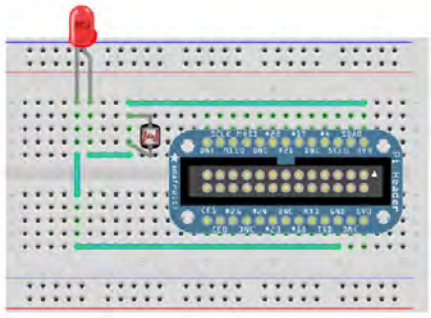


- elektrischer Widerstand, der sich mittels eines **Drehknopfes verändern lässt**
- der **mittlere Anschluss muss verwendet werden**, wenn man mit einem veränderlichen Widerstand arbeiten möchte
- wenn man nur die beiden äußeren Anschlüsse verwendet, arbeitet das Potentiometer mit maximalem Widerstand
- achtet beim Steckendes Potentiometers darauf, dass ihr nicht versehentlich zwei Pins via Steckbrett leitend verbindet!
- es ist empfehlenswert, das Potentiometer **in Reihe** mit einem **Schutzwiderstand** zu schalten, der den Strom begrenzt!

4.2.8 Beispielschaltungen und Lösungen

Im Foliensatz folgen nun wieder acht Beispielschaltungen. Analog zu Abschnitt 4.2.5 sollen sich die Schülerinnen und Schüler wieder Gedanken machen, ob ein geschlossener Stromkreis vorliegt, ob die LED durchbrennt und ob der Raspberry Pi kurzgeschlossen wird:



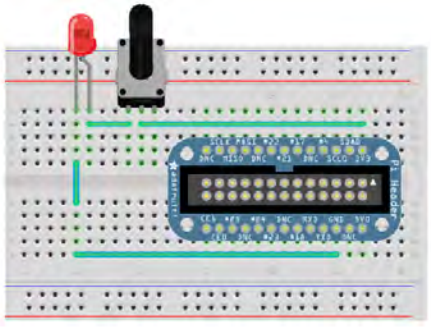


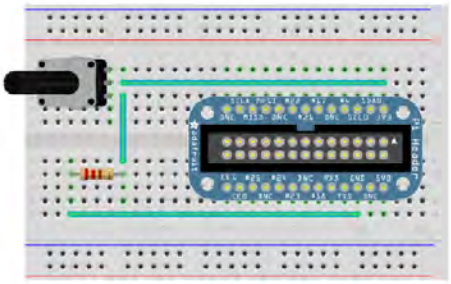
Linke Schaltung:

- geschlossener Stromkreis
- LED brennt nicht durch, da sich mit dem Fotowiderstand ein ausreichend großer Schutzwiderstand in der Schaltung befindet
- Raspberry Pi wird nicht kurz geschlossen

Rechte Schaltung:

- geschlossener Stromkreis
- LED brennt durch; der Fotowiderstand hat keinen Effekt, da alle Pins in der Spalte leitend miteinander verbunden sind
- Raspberry Pi wird kurzgeschlossen



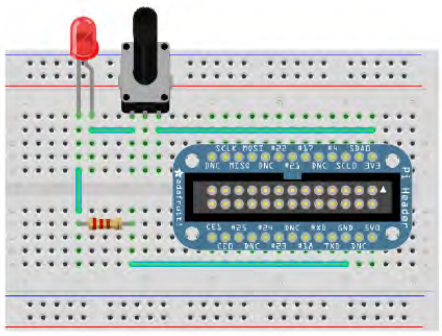


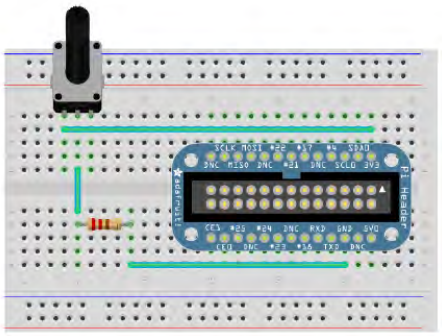
Linke Schaltung:

- geschlossener Stromkreis
- es besteht die Gefahr, dass die LED durchbrennt, wenn man den Widerstand des Potentiometers zu gering einstellt!
- Raspberry Pi kann kurz geschlossen werden

Rechte Schaltung:

- geschlossener Stromkreis!
- LED brennt nicht durch; Potentiometer hat allerdings keinen Effekt
- Raspberry Pi wird nicht kurzgeschlossen



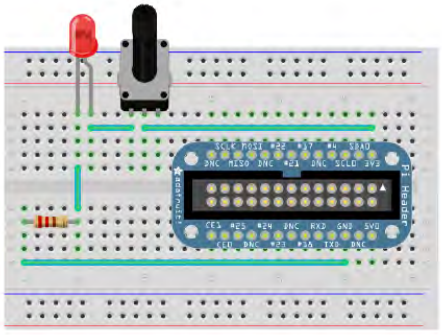


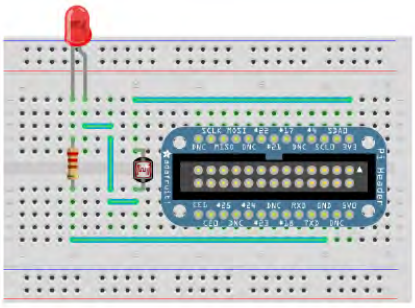
Linke Schaltung:

- geschlossener Stromkreis
- LED brennt nicht durch, leuchtet aber auch nicht, da beim Potentiometer die vollen 10 kOhm wirken, die nicht verstellt werden können (falsche Anschlüsse)
- Raspberry Pi wird nicht kurzgeschlossen

Rechte Schaltung:

- geschlossener Stromkreis
- LED brennt nicht durch; Potentiometer ist korrekt angeschlossen
- Raspberry Pi wird nicht kurzgeschlossen





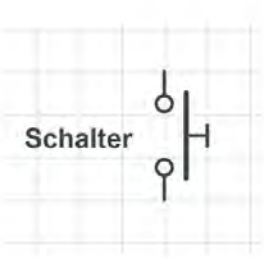
Linke Schaltung:

- geschlossener Stromkreis
- LED brennt nicht durch; Potentiometer ist korrekt angeschlossen
- Raspberry Pi wird nicht kurzgeschlossen

Rechte Schaltung:

- geschlossener Stromkreis
- LED brennt nicht durch
- Raspberry Pi wird nicht kurzgeschlossen

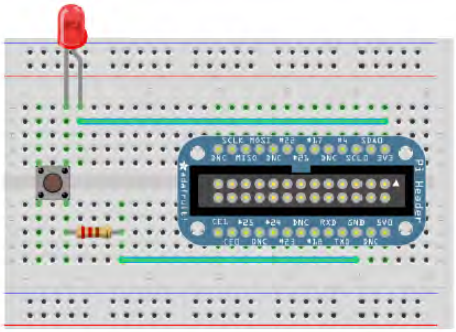
4.2.9 Schalter

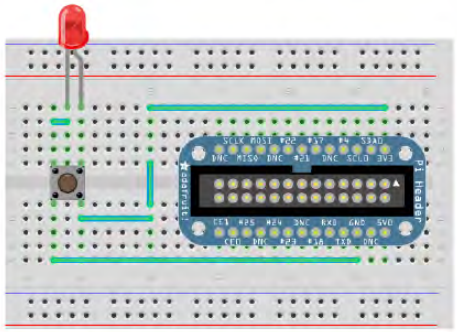


- ein offener Schalter hat „**unendlichen Widerstand**“
- ein geschlossener Schalter hat **keinen Widerstand**
- die jeweils farbig markierten Füße sind **immer leitend verbunden** (auch bei offenem Schalter)!!
- der Schalter wird eigentlich immer „über den Graben gesteckt“. Über den Graben kann man ihn nicht falsch orientieren, weil die Füße nicht reinpassen, wenn man ihn um 90 Grad dreht
- Achtet beim Stecken des Schalters auf das Breadboard und welche Pins leitend verbunden sind!

4.2.10 Beispielschaltungen und Lösungen

Im Foliensatz folgen nun vier Beispielschaltungen. Analog zu den Abschnitten 4.2.5 und 4.2.8 sollen sich die Schülerinnen und Schüler wieder Gedanken machen, ob ein geschlossener Stromkreis vorliegt, ob die LED durchbrennt und ob der Raspberry Pi kurzgeschlossen wird:



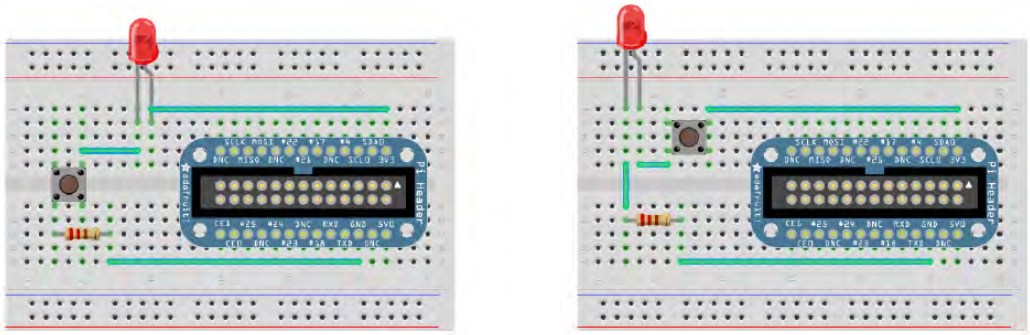


Linke Schaltung:

- geschlossener Stromkreis (auch wenn Schalter offen)
- LED brennt nicht durch, leuchtet allerdings ständig. Der Schalter hat keine Wirkung
- Raspberry Pi wird nicht kurzgeschlossen

Rechte Schaltung:

- geschlossener Stromkreis (auch wenn Schalter offen)
- LED brennt sofort durch, auch wenn Schalter nicht gedrückt wird
- Raspberry Pi wird kurzgeschlossen



Linke Schaltung:

- geschlossener Stromkreis, wenn Schalter geschlossen
- LED brennt nicht durch
- Raspberry Pi wird nicht kurzgeschlossen

Rechte Schaltung:

- geschlossener Stromkreis (auch wenn Schalter offen)
- LED brennt nicht durch, Schalter hat keine Wirkung
- Raspberry Pi wird nicht kurzgeschlossen

4.3 Aufgaben

4.3.1 Spannungsabfall untersuchen

In diesem Abschnitt verwenden wir den Raspberry PI nur als „eine Art Batterie“ mit zwei Anschlüssen (3,3 Volt und Erde). Dabei geht es darum, dich mit den ersten Bauelementen, dem Breadboard und Begriffen aus der Physik vertraut zu machen. Bevor wir etwas mehr zu Spannung und Spannungsabfall sagen, sollst du erst einmal selbst damit experimentieren:

4.3.1.1 Leuchtende LED

Zunächst wollen wir eine Leuchtdiode (LED) leuchten lassen.
Die grafische Darstellung der elektrischen Schaltung, die wir verwenden, liest man am besten von oben nach unten. Man folgt der Leitung von 3,3 V - der Physiker spricht auch vom Potential 3,3 V (eine Art Niveau) - über die Leuchtdiode weiter zum Widerstand, bis man schließlich an der Erde (oder Masse) ankommt.

Betrachten wir zunächst den linken Schaltplan:
??

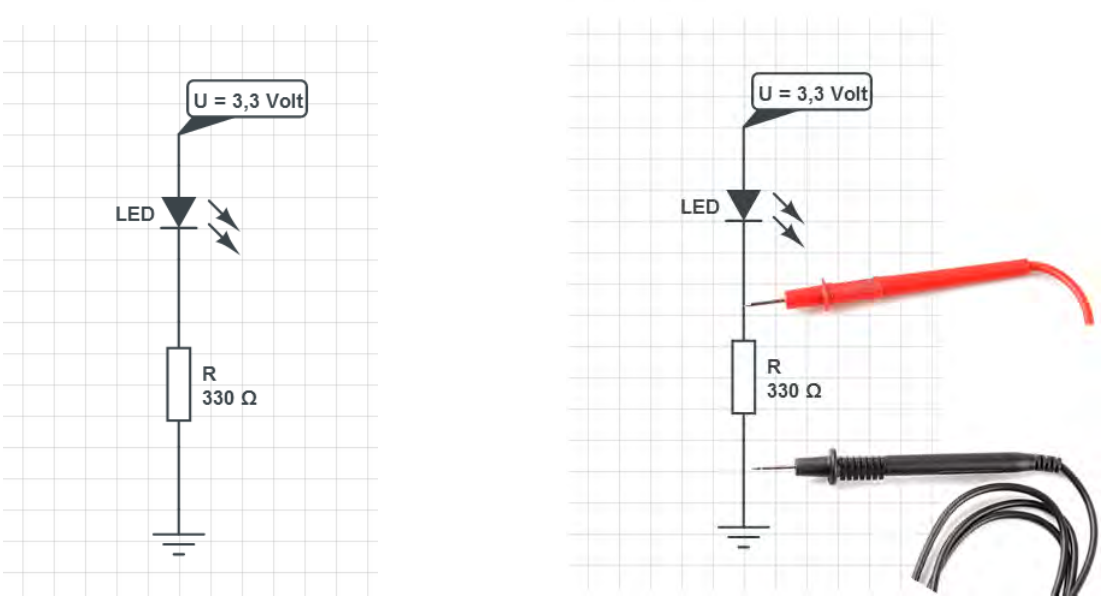


Abbildung 4.1: Schaltplan 1.1

Mit einem Spannungsmessgerät (Voltmeter) kann man herausfinden, welche Spannung am Widerstand abfällt. Dazu verbindet man das schwarze Messkabel mit Masse (Com-Anschluss am Voltmeter) und das rote Kabel mit dem in obigen Schaltplan eingezeichneten Messpunkt.

Hinweis: Ein Voltmeter hat erfreulicherweise einen sehr großen Widerstand. Misst man eine Spannung, so „zapft man quasi keinen Strom ab“.

Setze nun obigen Schaltplan um und notiere den Spannungsabfall am Widerstand, den du mit dem Voltmeter gemessen hast!
Ein mögliches Umsetzungsbeispiel sieht wie folgt aus. Lies dir unbedingt untenstehende Hinweise durch, bevor du mit dem Erstellen deiner Schaltung anfängst!

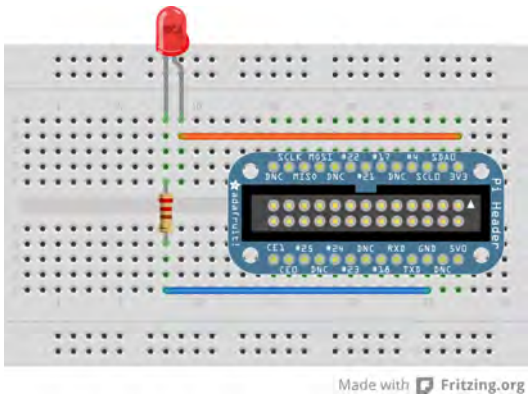
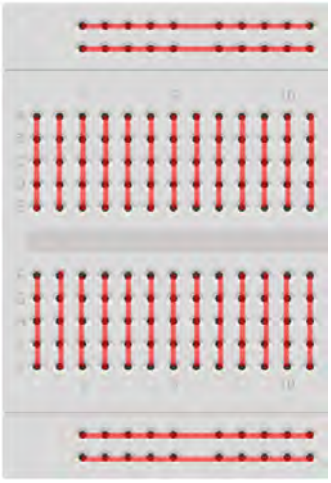


Abbildung 4.2: Umsetzungsbeispiel für den Schaltplan 1.1

Wichtige Hinweise:

- Die Pins des Steckbretts sind nach dem rechts abgebildeten Prinzip leitend verbunden: Man spricht auch von horizontalen Kontaktfeldern (in der Abbildung oben und unten angebracht) und vertikalen Kontaktfeldern.
  Die Lage der Kontaktfelder kann je nach Steckbrett variieren. Die horizontalen Kontaktfelder sind häufig durch eine rote und eine blaue durchgängige Linie auf dem Streckbrett gekennzeichent. Es bietet sich an, die blaue Linie für den Anschluss an Erde (man sagt auch Masse) zu verwenden.

- Bei der **LED** macht es einen Unterschied, wie herum sie in die Schaltung eingesetzt wird - Dioden *lassen den Strom nur in einer Richtung durch*. Die Minus-Seite lässt sich an dem LED-Plastikgehäuse erkennen - sie wird durch eine Abflachung markiert.[a]
  Die Minus-Seite (Kathode) *muss in Richtung Ground* (GND) zeigen. Die positive Seite nennt man Anode.



- Eine LED ist immer *in Reihe mit* einem **Schutzwiderstand zu schalten**, der den Strom begrenzt. Fließt ein zu großer Strom durch die LED, so „brennt sie durch“.
  Mit einem Messgerät kannst du testen, ob eine LED kaputt ist. Frag im Zweifel deinen Übungsleiter um Hilfe!

[a] Alternativ kann man sich auch merken, dass der längere Fuß der Plus-Pol ist - es dauert länger, ein Plus zu zeichnen, als ein Minus.

Weitere Fragen:

- ▶ Wie misst man den Spannungsabfall an der LED? Wie groß ist dieser?
- ▶ Was fällt dir auf, wenn du den Spannungsabfall am Widerstand und an der LED addierst?
- ▶ Wie könnte man den Spannungsabfall an beiden Steckelementen mit dem Voltmeter messen?

4.3.1.2 Fotowiderstand (LDR)

Ersetze nun die LED durch einen Fotowiderstand - LDR. Die beiden Anschlüsse des LDR müssen nicht unterschieden werden.

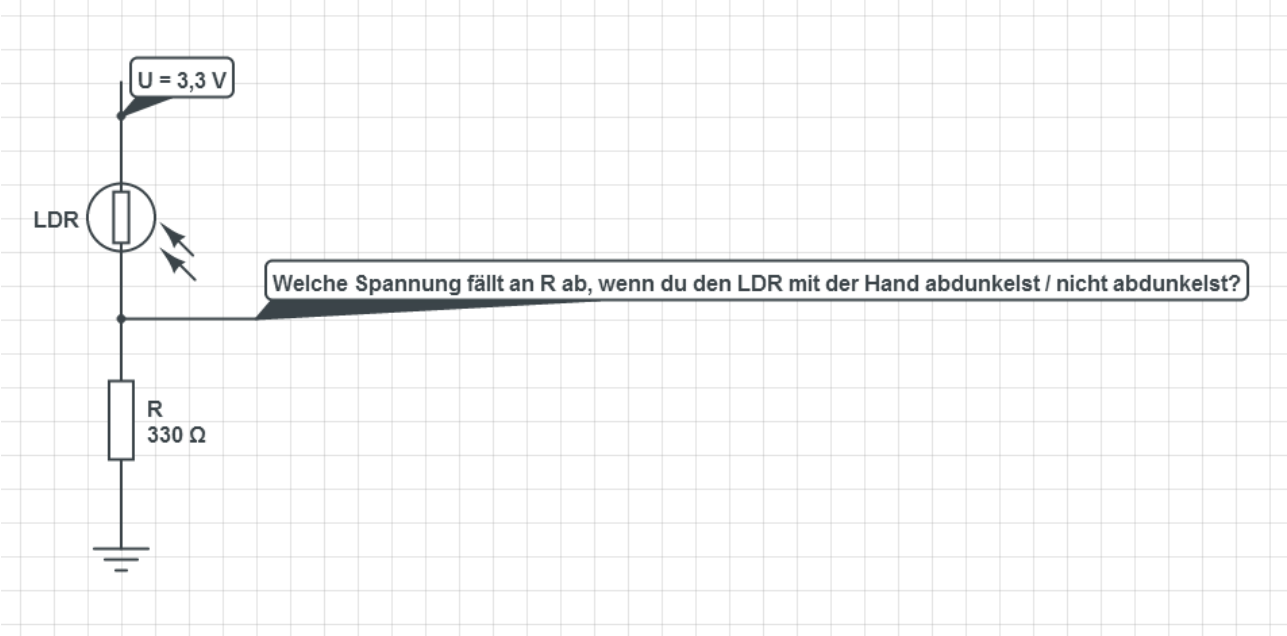


Abbildung 4.3: Schaltplan 1.2

Aufgaben

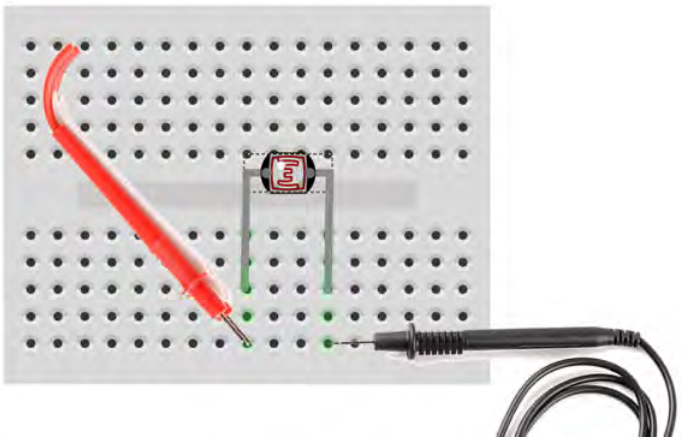
- Versuche die Funktionsweise des LDR mit eigenen Worten zu beschreiben!
- Welchen Spannungsabfall misst du am Widerstand?
- Vergleiche den Spannungsabfall am Widerstand mit dem bei der Aufgabe vorher mit der LED. Was fällt auf? Hast du eine Erklärung?

Widerstände messen

Mithilfe des Multimeters kann man auch Widerstände von einzelnen Steckelemente messen. Wir möchten gerne die Widerstände des LDR und des 330 Ohm-Widerstands[1] messen. Dazu steckst du die Steckelemente an beliebiger Stelle in das Breadboard, wobei die beiden „Füße“ eines Steckelements nicht leitend verbunden sein sollen. Das Multimeter musst du noch auf das Symbol Ω (Ω steht für Ohm und somit für die Einheit des Widerstandes) umschalten.

Den Widerstand des LDR misst du wie folgt:

[1] Bei dem wir hoffentlich auch 330 Ohm messen ☺



Dunkle während du den Widerstand des LDR misst, diesen ab bzw. nicht ab. Was fällt dir auf?

Vergleiche das Verhältnis der Widerstände von LDR und Widerstand mit dem Verhältnis der Spannungsabfälle an den beiden Bauelementen. Was fällt dir auf?

4.3.1.3 Potentiometer

Ersetze nun den LDR durch ein Potentiometer. Verwende dabei den mittleren Anschluss und einen äußeren Anschluss (egal welchen). Ersetze zudem den 330 Ohm Widerstand durch einen 3,3 kOhm Widerstand:

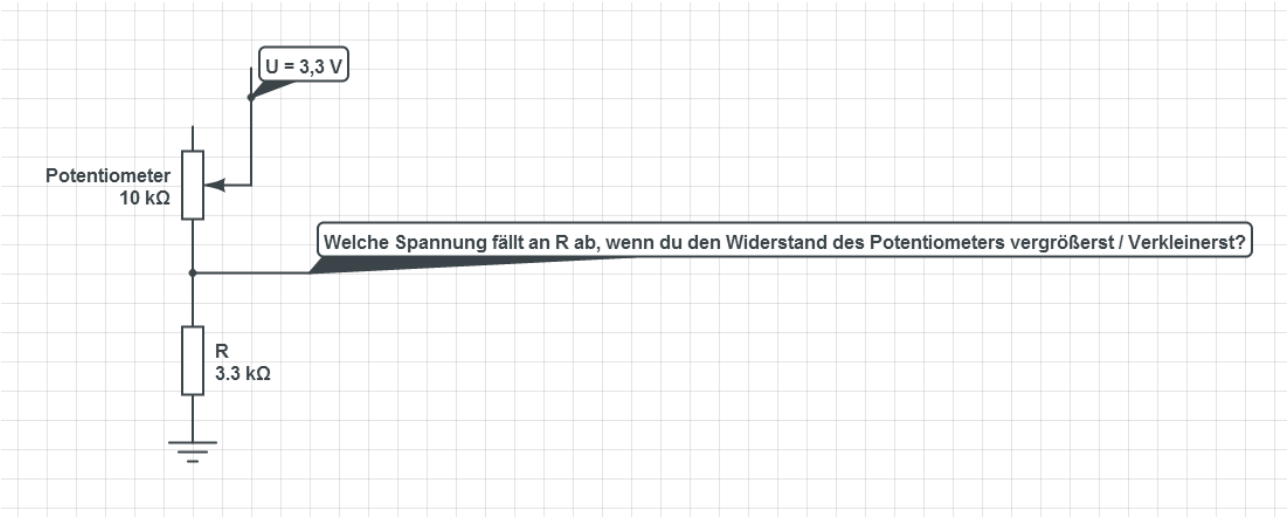


Abbildung 4.4: Schaltplan 1.3

Widerstände messen

Wir möchten wie bei der Aufgabe vorher nun auch wieder die Widerstände der Bauelemente messen und deren Verhältnis mit dem Verhältnis der Spannungsabfälle vergleichen. Stelle dazu das Potentiometer zunächst auf einen beliebigen Wert ein und miss den Spannungsabfall an je beiden Elementen - notiere dir die beiden Werte. Anschließend sollst du wie vorher den Widerstand des Potentiometers (Wichtig: Drehknopf nicht verändern!) und des 3,3 kOhm Widerstands. Notiere dir auch diese Werte! Vergleiche wieder die beiden Verhältnisse! Berechne und vergleiche die Verhältnisse noch für eine andere Einstellung des Drehknopfes!

4.3.1.4 Druckwiderstand

Ersetze das Potentiometer durch einen druckempfindlichen Widerstand. Messe den Spannungsabfall am Widerstand, wenn du den Druckwiderstand drückst bzw. nicht drückst. Versuche nun, ohne konkret den Widerstand am Druckwiderstand zu messen, herauszufinden, ob der Widerstand am Druckwiderstand größer ist, wenn man ihn drückt oder nicht drückt.

4.3.1.5 Temperaturempfindlicher Widerstand (NTC)

Ersetze den Druckwiderstand durch einen NTC-Widerstand (engl.: *Negative Temperature Coefficient Thermistor*). Versuche auch hier, ohne konkret den Widerstand zu messen, herauszufinden, ob der NTC einen höheren Widerstand hat, wenn du ihn mit der Hand erwärmst, oder in kaltes Wasser tauchst.

4.3.1.6 Schalter

Ersetze das Potentiometer durch einen Schalter. Verwende zwei Anschlüsse, die sich diagonal gegenüberliegen. Wie ändert sich der zwischen M und Erde gemessene Spannungsabfall, wenn du den Schalter schließt / öffnest? Welchen Widerstand hat ein Schalter also formal, wenn er offen ist bzw. wenn er geschlossen ist?

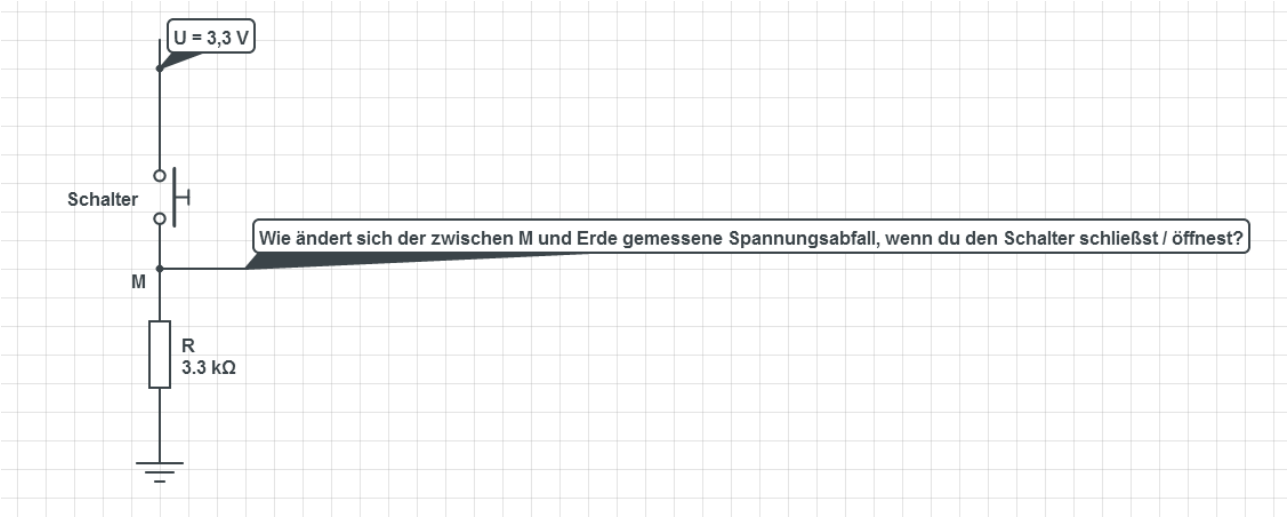


Abbildung 4.5: Schaltplan 1.4

4.3.2 Spannungen via Programm an- bzw. nicht anlegen

Bislang haben wir den Raspberry Pi nur als Stromquelle genutzt und mit der konstanten Spannung 3,3 V gearbeitet. Nun wollen wir mit einem Programm an einen Pin (wir wählen Pin 17) selbst Spannung anlegen bzw. eben nicht anlegen. Verwende dazu den Schaltplan 1.1, nur dass du nicht die konstante Spannung 3,3 V verwendest, sondern das Steckkabel mit Pin 17 verbindest:

Wir verwenden die Programmiersprache Python, um an Pin 17 Spannung an- oder abschalten zu können. Dazu haben wir in Abschnitt 4.1 die Python-Library und den Editor Geany installiert.

Zunächst importieren wir das `RPi GPIO` Modul, stellen die Pin-Konfiguration auf BCM[2] und legen fest, dass wir an den Pin mit der Nummer 17 eine Spannung anlegen wollen, die dann unsere LED zum Leuchten bringen soll:

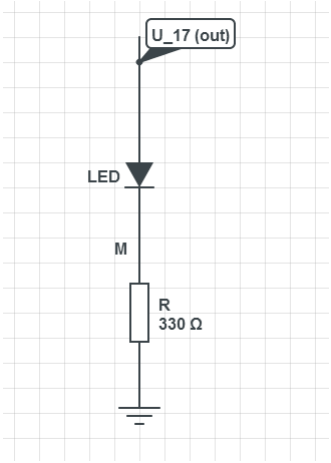


Abbildung 4.6: Schaltplan 2

```
1 # Importieren des RPi.GPIO Moduls
2 import RPi.GPIO as GPIO
3 # Initialisieren der Variable ledPin mit dem Wert 17
4 ledPin = 17
5 # Einstellen der Pin-Konfiguration auf BCM
6 GPIO.setmode(GPIO.BCM)
7 # Festlegen des Pin 17 als Ausgang
8 GPIO.setup(ledPin,GPIO.OUT)
```

Jetzt können wir an Pin 17 eine Spannung anlegen / nicht anlegen. Der Raspberry Pi hat nur digitale Ein- und Ausgang-Pins, d.h. er kann nur `HIGH` (Spannung ist angelegt) und `LOW` (Spannung ist nicht angelegt) unterscheiden.

4.3.2.1 Blinken mit fester Anzahl

Wir möchten nun 5 mal hintereinander Spannung anlegen, zwei Sekunden warten und Spannung wieder abschalten. Um zwei Sekunden warten zu können, müssen wir noch das `time` Modul importieren. Dies schaffst du mit der Codezeile:

```
1 import time
```

Nun können wir obiges Szenario in Python umsetzen. `time.sleep(x)` hält das Programm dabei für `x` Sekunden an:

```
1 for i in range(5):
2     GPIO.output(ledPin, GPIO.HIGH)  # Spannung anlegen
3     time.sleep(2)                   # 2 Sekunden warten
4     GPIO.output(ledPin, GPIO.LOW)   # Spannung nicht anlegen
5     time.sleep(2)                   # 2 Sekunden warten
```

[2] Auf diese Art und Weise ist der Cobbler verkabelt

Teste dein Programm nun, ob es funktioniert!

Nachdem das Programm fertig abgearbeitet wurde, bleibt ein schwarzes Fenster zurück. Beende dieses mit `Strg + C`; machst du das nicht, kann es sein, dass beim nächsten Programmstart Fehlermeldungen auftreten!

Du fragst dich möglicherweise, warum wir zwischen dem An- und Abschalten der LED jeweils 2 Sekunden warten. Das kannst du schnell rausfinden, indem du beide Zeilen bzw. je eine Zeile mit einem # auskommentierst und schaust, was nun bei Ausführung deines Programms passiert.

4.3.2.2 Endlos Blinken

Die Schaltung bleibt unverändert.

Verwende anstatt der Wiederholung mit fester Anzahl eine unendliche Wiederholung. Das Programm kann wieder via `Strg + C` gestoppt werden - ein Keyboard Interrupt sorgt dann für den Abbruch des Programms.
Bei einem solchen Abbruch muss noch „aufgeräumt“ werden. Alle Pins sind auf ihren Standardwert zurückzusetzen. Macht man dies nicht, geht zwar nichts kaputt, aber nach jedem Programmstart erhält man eine Fehlermeldung im Terminal. Dies leistet der Befehl `GPIO.cleanup()`.

Hinweis: Eine endlose Wiederholung mit anschließendem „Aufräumen“ hat in Python folgende Syntax:

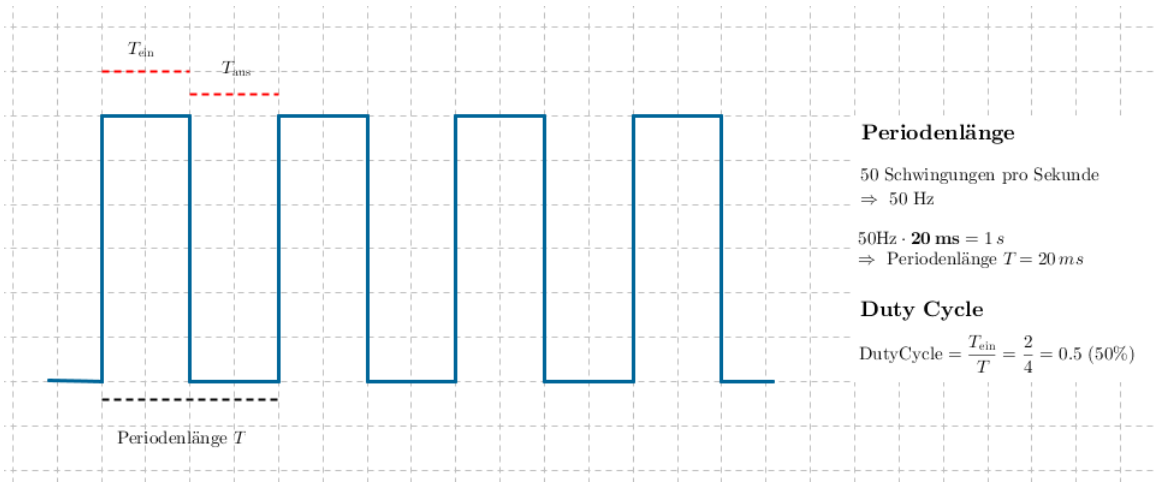
```
1 try:
2     while True:
3         # Code, der endlos wiederholt werden soll
4 finally:
5     GPIO.cleanup()
```

4.3.2.3 Pulsweitenmodulation - Blinkprozess starten

Die Schaltung bleibt unverändert. Probier folgendes Programm aus:

```
1  import RPi.GPIO as GPIO
2  ledPin = 17
3  GPIO.setup(ledPin, GPIO.OUT) # bis hierher wie gehabt
4
5  # 2 Hz -> 2 Schwingungen pro Sekunde -> LED geht nach 0.5s wieder an
6  blinken = GPIO.PWM(ledPin, 2)
7  # 50 Prozent der Zeit ist die LED an. Probiere hier andere Werte aus!
8  blinken.start(50) # Blinkprozess starten
9  # Programm kann durch Eingabetaste beendet werden
10 raw_input()
11 blinken.stop()
12 GPIO.cleanup()
```

Kann man die LED auch schneller / langsamer blinken lassen?



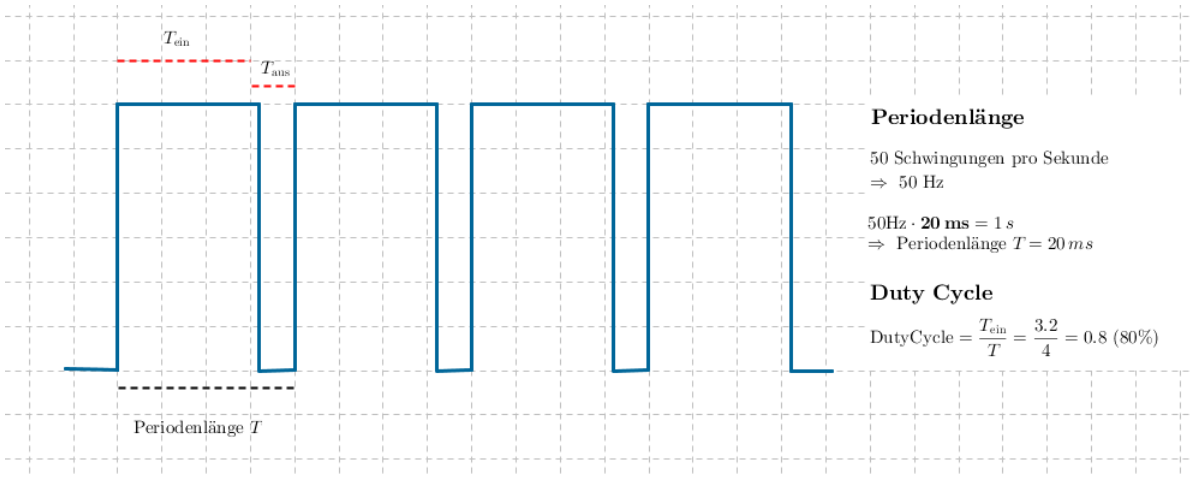
Möchte man 50 Schwingungen pro Sekunde erreichen, muss man den Parameterwert in folgendem Befehl ändern:

```
1 # 50 Hz -> 50 Schwingungen pro Sekunde -> LED geht nach 20ms wieder an
2 blinken = GPIO.PWM(ledPin, 50)
```

Welchen Einfluss hat der Parameterwert x in folgendem Befehl?

```
1 blinken.start(x)
```

Bei einem Wert $x = 50$ ist die LED pro Schwingung die Hälfte der Zeit an und die Hälfte der Zeit aus. Bei einem Wert $x = 80$ ist die LED 80% der Zeit an und 20% der Zeit aus. Folgende Abbildung verdeutlicht den Einfluss des Parameters auf die Pulsweitenmodulation noch einmal:



Kann man die Unterschiede zwischen $x = 50$ und $x = 80$ auch mit dem Voltmeter detektieren? Miss für beide Einstellungen die Spannung U_{17}. Was fällt dir auf?

Mit folgendem Befehl kannst du während der Ausführung den DutyCycle der Pulsweitenmodulation ändern::

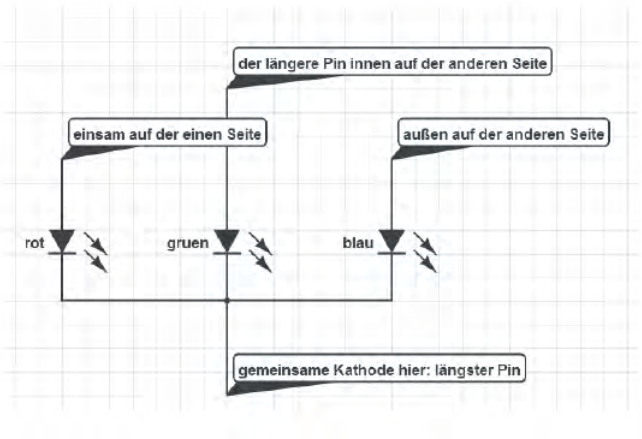
```
1 blinken.ChangeDutyCycle(20) # Setzt den DutyCycle auf 20 Prozent
```

Für was könnte man Pulsweitenmodulation verwenden?

4.3.2.4 PWM und RGB-LED - Blinken in Regenbogenfarben

Bei dieser Aufgabe arbeiten wir zum ersten mal mit der RGB-LED. Dabei geht es darum, den Aufbau dieser besonderen LED kennenzulernen und sich noch weiter mit der Pulsweitenmodulation zu beschäftigen.

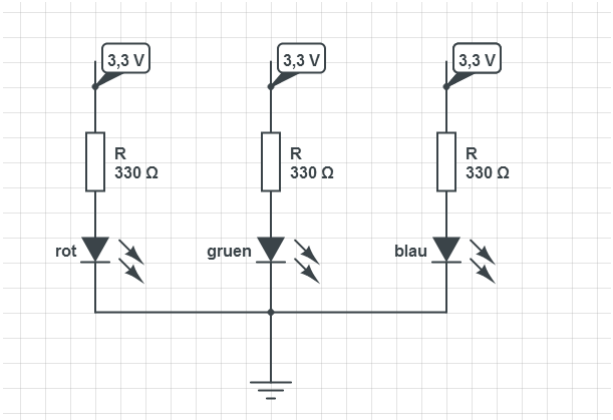
Aufbau einer RGB-LED



Im Gegensatz zu einer normalen LED hat eine RGB-LED nicht zwei sondern vier Füße. Eine RGB-LED kann in drei verschiedenen Farben leuchten. Dabei stehen drei Füße für die Farben rot, grün und blau. Der vierte Fuß ist die gemeinsame Kathode der drei LEDs - die gemeinsame Erdung. In obiger rechter Abbildung ist der Aufbau der RGB-LED dargestellt; insbesondere welcher Fuß für welche Farbe bzw. für die Kathode steht.

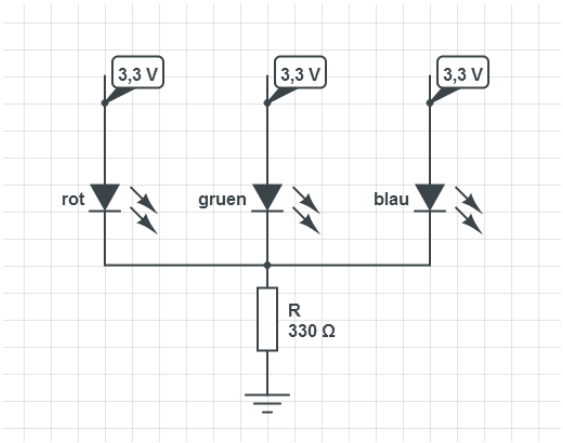
Alle Farben zum Leuchten bringen

Zunächst wollen wir alle Farben einfach mal mit der konstanten Spannung des Pi zum Leuchten bringen. Setze dazu folgenden Schaltplan um und schließe abwechselnd die drei Stromkreise:



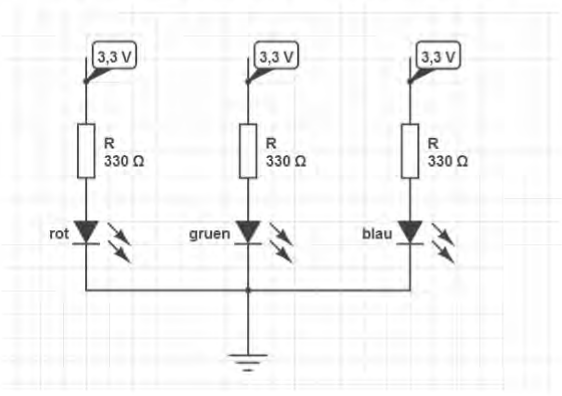
Hinweis: Vergiss auf keinen Fall die drei Widerstände! Ansonsten geht die RGB-LED kaputt!

Vielleicht fragst du dich, warum man nicht einfach nur einen Widerstand verwendet und diesen zwischen der gemeinsamen Kathode und `GND` in die Schaltung einfügt:

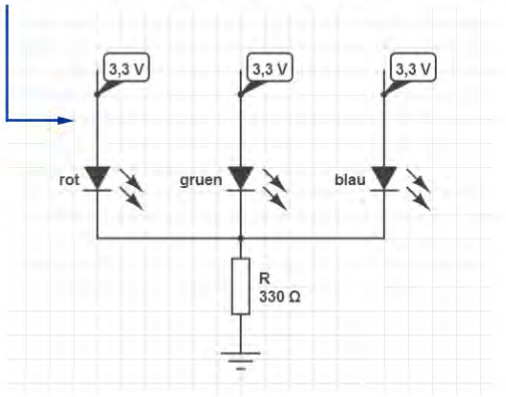


Sollte man besser drei Widerstände verwenden, oder nur einen an der gemeinsamen Kathode? Ist das egal? Ausprobieren!

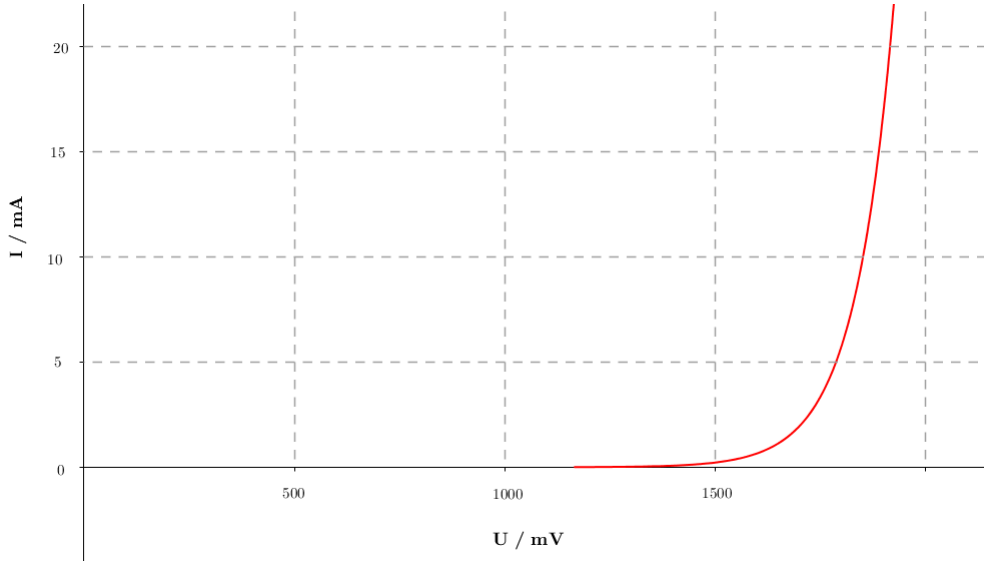
Welche Spannungen fallen hier jeweils an den drei LEDs ab?



Was passiert, wenn man hier den Stecker zieht?
Welche Spannung fällt vorher / nachher an der grünen / blauen LED ab?



Kennlinie einer roten LED



Was will uns so ein Graph sagen?

(1) Der Widerstand einer LED ist alles andere als konstant! [3]

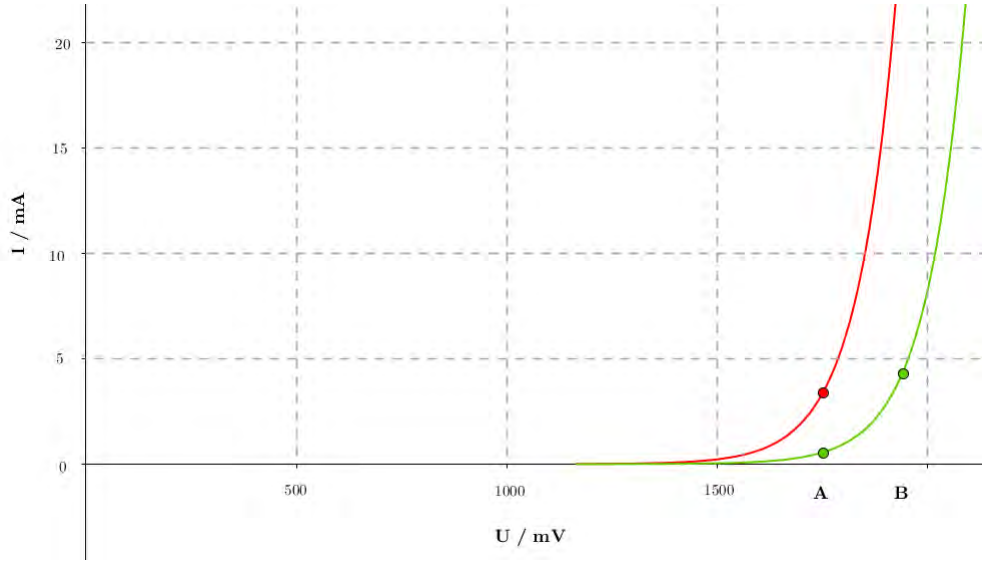
(a) Legt man eine „kleine“ Spannung an, hat die LED einen sehr großen Widerstand: $R = \frac{U}{I}$

Kurz: Wenn die LED nicht leuchtet, dann ist ihr Widerstand groß!

(b) Legt man eine „größere“ Spannung an (LED beginnt zu leuchten), dann nimmt der Widerstand rapide ab.

Kurz: Je heller die LED leuchtet, desto kleiner ist ihr Widerstand - bis sie zerstört wird. Eine relativ kleine Erhöhung einer Spannung kann eine LED zerstören!

(2) Kennlinie einer roten und einer grünen LED:

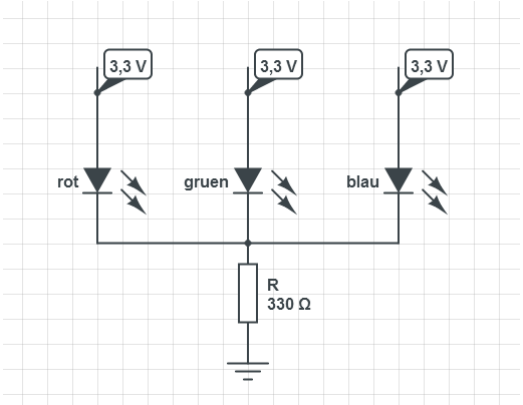


Vergleich der beiden Graphen?

(a) Wenn an der roten und an der grünen LED eine Spannung A anliegt, dann wird nur die rote LED nett leuchten

(b) Wenn an der roten und an der grünen LED eine Spannung B anliegt, dann ist die rote LED zerstört und die grüne LED leuchtet schön hell

(3) Welche Spannungen fallen jeweils an den LEDs ab?



Es fällt an allen drei LEDs die gleiche Spannung ab ($\approx 1,87\,\mathbf{V}$)

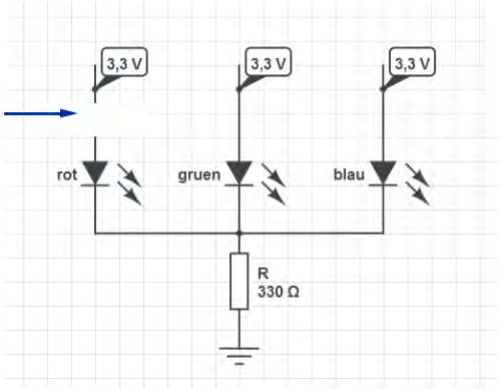
Brennt die rote LED durch?

Nein, der Schutzwiderstand begrenzt den Gesamtstrom auf $I = \frac{U}{R} = \frac{3,3\,\mathrm{V}}{330\,\Omega} = 10\,\mathrm{mA}$

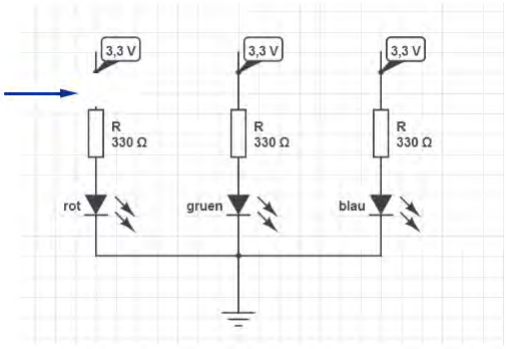
Da rot nicht durchbrennt, wird grün auch nicht schön leuchten!

[3] Nebenbemerkung: Von DEM Widerstand einer LED zu sprechen ist also „Schmarrn“.

(4) Was passiert, wenn man hier den Stecker zieht? Welche Spannung fällt vorher / nachher an der grünen / an der blauen LED ab? Messen!



(4) Was passiert, wenn man hier den Stecker zieht? Welche Spannung fällt vorher / nachher an der grünen / an der blauen LED ab? Messen!



Hier ändert sich nichts an $U_{\mathrm{LED_{grün}}}$ bzw. $U_{\mathrm{LED_{blau}}}$!

Welche Spannungen fallen hier jeweils an den drei LEDs ab? Verschiedene! Das ist „gut“ für die verschiedenfarbigen LEDs!

Pulsweitenmodulation und RGB-LED

Experimentiere mit folgendem Programm, indem du insbesondere die Werte an den kommentieren Stellen vergrößerst / verkleinerst! Versuche soviele Farben wie möglich zu erzeugen!

```
1 from RPi.GPIO import *
2 import time
3
4 setmode(BCM)
5 rot = 4
6 gruen = 17
7 blau = 22
8
9 setup(rot, OUT)
```

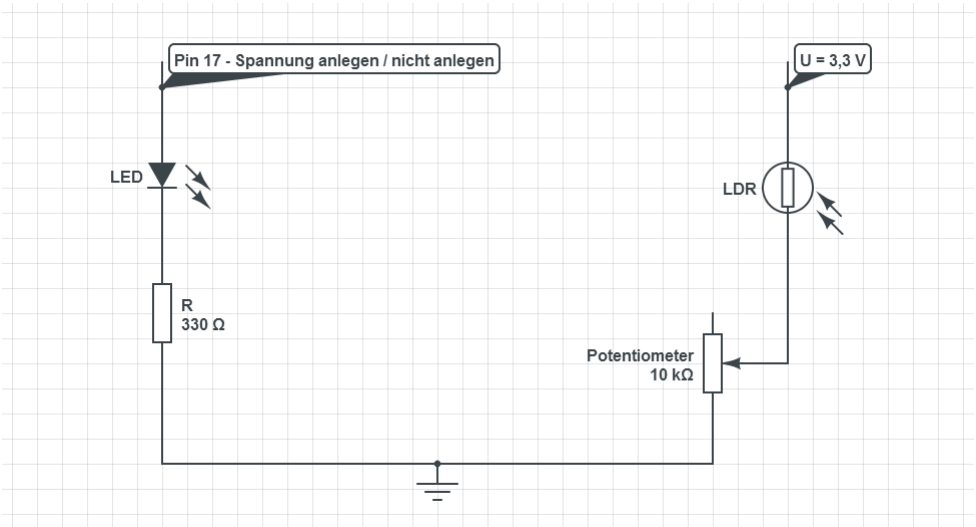
```
setup(gruen, OUT)
setup(blau, OUT)

rotblinken = PWM(rot, 1) # 1 Hz - andere Werte ausprobieren
gruenblinken = PWM(gruen, 1) # 1 Hz - andere Werte ausprobieren
blaublinken = PWM(blau, 1) # 1 Hz - andere Werte ausprobieren

rotblinken.start(50) # 50 Prozent der Zeit ist die LED an - andere Werte ausprobieren
gruenblinken.start(20) # 20 Prozent der Zeit ist die LED an - andere Werte ausprobieren
blaublinken.start(80) # 80 Prozent der Zeit ist die LED an - andere Werte ausprobieren

raw_input()
rotblinken.stop()
gruenblinken.stop()
blaublinken.stop()

cleanup()
```

4.3.3 Spannungen via Programm messen - LED steuern

Bisher haben wir die GPIO-Pins des Raspberry Pi nur dazu verwendet, um via Python-Programm Spannung anlegen zu können. In dieser Aufgabe wollen wir zum ersten mal mit einem Pin Spannung, die zwischen einem Messpunkt M und `GND` (Erde) abfällt, auch messen!

4.3.3.1 Abdunkeln detektieren

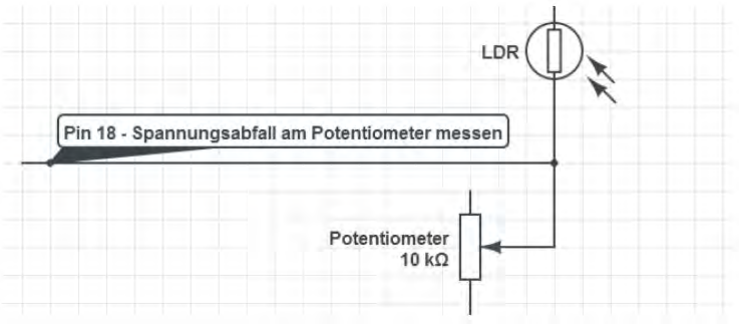
Setze unten abgebildeten Schaltplan in eine Schaltung um. Wundere dich nicht, dass die LED nicht leuchtet - da kann sie am Anfang noch gar nicht!



Voraufgabe:

- Stelle das Potentiometer so ein, dass am Potentiometer ein Spannungsabfall $\geq 1,4\,\mathrm{V}$ gemessen wird, falls der Lichtwiderstand nicht abgedunkelt wird und $\leq 1,2\,\mathbf{V}$, falls der Lichtwiderstand abgedunkelt wird.

Den Spannungsabfall können wir, ähnlich wie mit dem Voltmeter, auch mit einem Pin des Raspberry Pis messen. Dazu müssen wir diesen Pin an geeigneter Stelle mit unserer Schaltung verbinden:



Ergänze deine Schaltung entsprechend!

Auslesen, ob abgedunkelt wird oder nicht

Wir haben dir bereits die Rohfassung des Programms vorgegeben:

```
1  # Importieren des RPi.GPIO Moduls
2  from RPi.GPIO import *
3
4  # Einstellen der Pin-Konfiguration auf BCM
5  setmode(BCM)
6
7  # Initialisieren der Variable ledPin und messPin
8  ledPin = 17
9  messPin = 18
10
11 # Festlegen des Pin 17 als Ausgang und Pin 18 als Eingang
12 setup(ledPin, OUT)
13 setup(messPin, IN)
```

Verwende eine unendliche Wiederholung (denke dabei auch an den `try/finally`-Block) und lass dir zunächst den Wert ausgeben, der am Eingangs-Pin 18 gemessen wird[4], falls der Lichtwiderstand abgedunkelt wird bzw. nicht abgedunkelt wird. Notiere dir die Werte!

4.3.3.2 LED leuchtet, LED leuchtet nicht

Ändere dein Programm nun so ab, dass die LED leuchtet, wenn der Lichtwiderstand abgedunkelt wird - ansonsten soll die LED nicht leuchten.

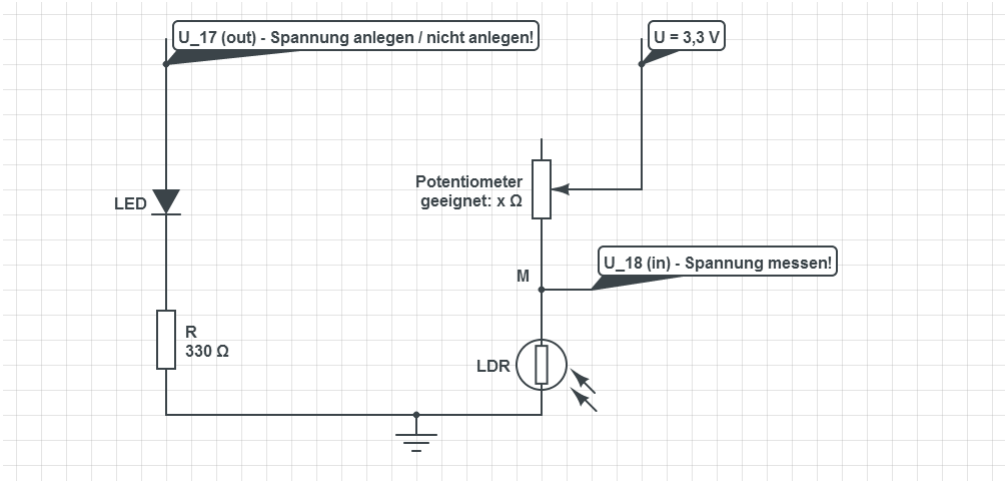
Hinweise:

- der Mess-Pin 18 hilft dir, entscheiden zu können, ob der Lichtwiderstand abgedunkelt ist oder nicht (vgl. Teilaufgabe 4.3.3.1)
- nun musst du nur noch geeignet an Pin 17 Spannung anlegen bzw. nicht anlegen

[4] Mit `input(x) kannst du den Wert auslesen, der an Eingang x gemessen wird`

4.3.3.3 LDR und Potentiometer vertauschen

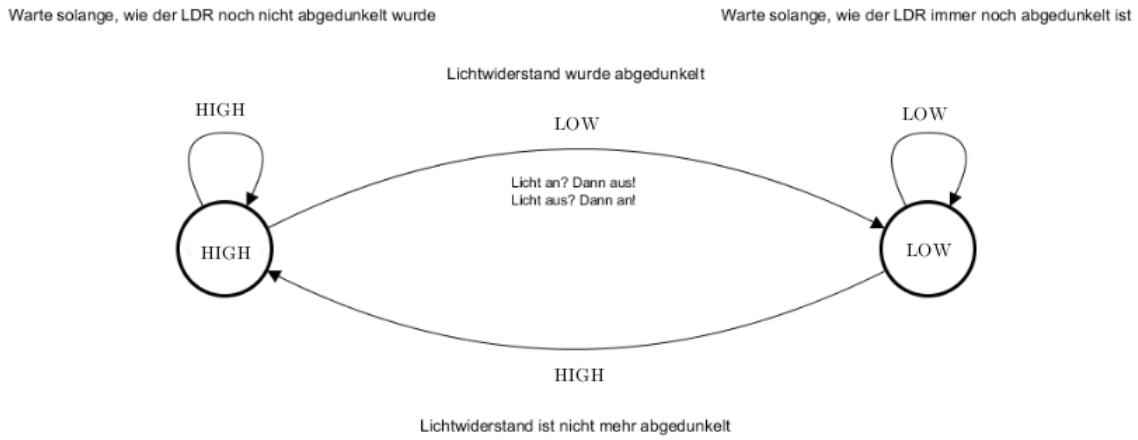
Vertausche Potentiometer und LDR entsprechend nachfolgendem Schaltplan, ohne die Einstellung des Potentiometers zu ändern.
Wann leuchtet die LED?



4.3.3.4 Lichtschalter

Wir arbeiten bei dieser Aufgabe wieder mit dem ursprünglichen Schaltplan aus Aufgabe 4.3.3.1.
Verändere dein Programm so, dass sich die LED durch Abdunkeln des LDR „einschalten“ lässt und auch weiterhin leuchtet, wenn man den LDR nicht mehr abdunkelt. Wenn man den LDR nun erneut abdunkelt, soll die LED „ausgeschaltet“ werden, usw.

Du kannst diese Aufgabe beispielsweise durch das Einführen von weiterer Variablen lösen, oder dich an folgendem Diagramm (Verwendung von Haltepunkten) orientieren:

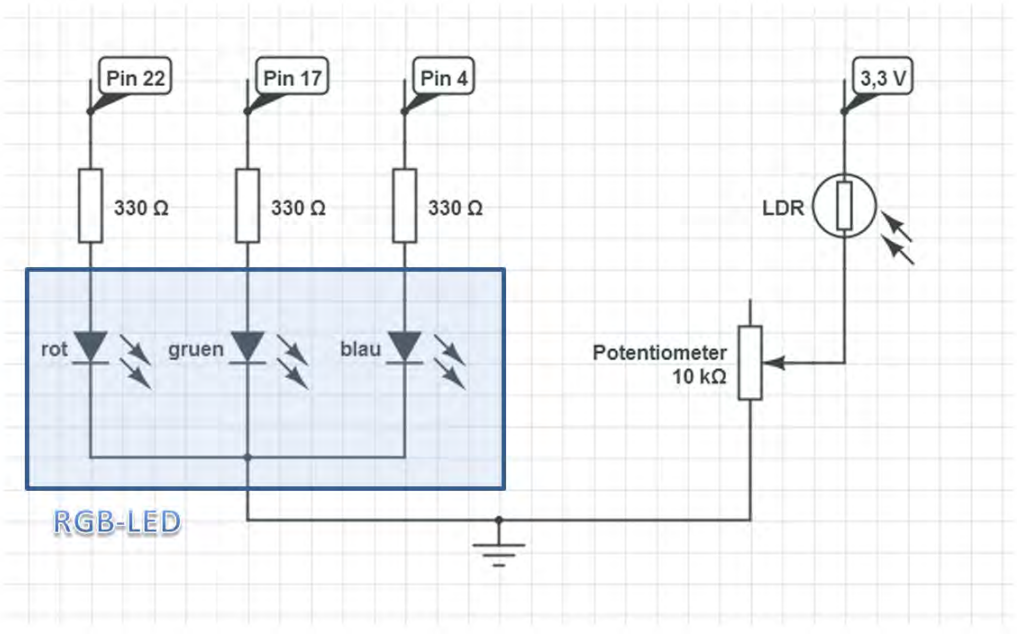


Leere `while`-Wiederholungen haben in Python folgende Syntax:

```
1 while input(messPin) == HIGH:
2     pass
```

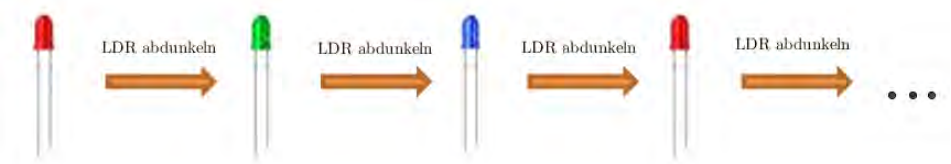
4.3.3.5 RGB-LED durchschalten

Ersetze die LED im bisherigen Schaltplan wie folgt durch eine RGB-LED:



Aufgabe 1 - Farben „durchschalten“

Anfangs soll die RGB-LED rot leuchten. Wenn du den LDR abdunkelst, soll die RGB-LED grün leuchten - und zwar solange, bis du den LDR erneut abdunkelst. Ist dies der Fall, soll die RGB-LED blau leuchten.



Setze diese Aufgabe in ein Programm um - überlege dir zunächst, welche Variablen du benötigst und welche Pins du als Ausgänge bzw. Eingänge festlegen musst. Du kannst dich dabei natürlich an den vorherigen Aufgaben orientieren.

Aufgabe 2 - Farbwechsel

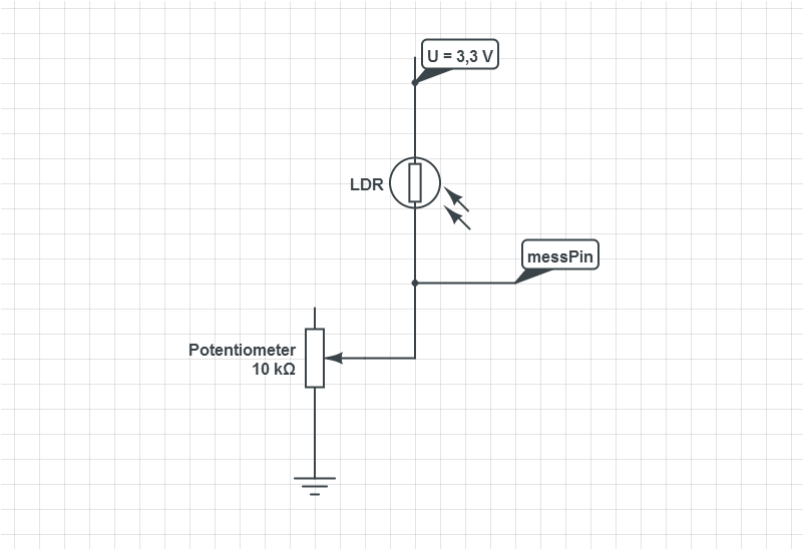
Ähnliche Aufgabenstellung wie in Aufgabe 1 - nur sollen die Farben nun ständig wechseln, **solange** du den LDR abdunkelst. Wenn du den LDR nicht mehr abdunkelst, soll die aktuelle Farbe leuchten.
Diesmal sollen die Farben also via Abdunkeln nicht mehr sequentiell durchgeschaltet werden, sondern sich während dem Abdunkeln ständig verändern.

4.3.4 Kondensator

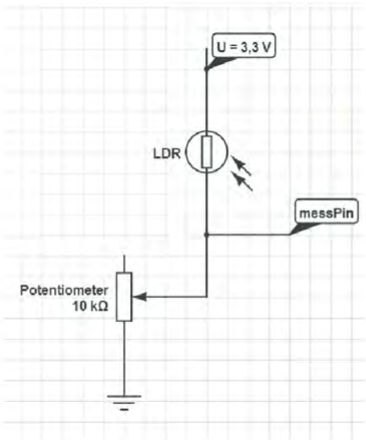
In diesem Abschnitt lernst du ein weiteres Steckelement kennen - den Kondensator.
Bevor wir den Kondensator verwenden, befassen wir uns mit einem Problem, welches wir u.a. auch mit einem Kondensator lösen können.

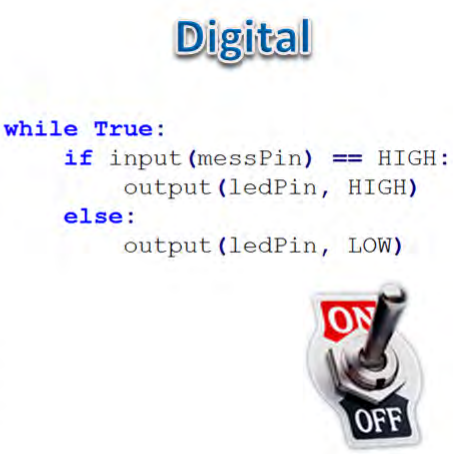
Mehr als hell / dunkel messen

Wir haben bereits mehrmals die Problematik angesprochen, dass der Raspberry Pi nur mit digitalen Ein- und Ausgängen ausgestattet ist, d.h. wir können nur `HIGH` oder `LOW` anlegen bzw. messen.
Betrachten wir folgende Schaltung, die wir schon öfter verwendet haben:

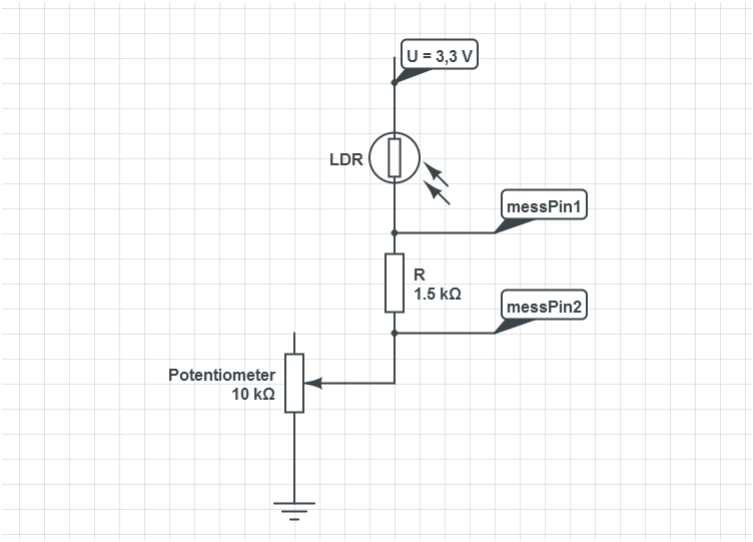


Schön wäre es, mehrere Helligkeitsstufen messen zu können und nicht nur hell oder dunkel. Warum wäre dies ohne Probleme möglich, wenn der Raspberry Pi analoge Eingänge hätte?





Betrachte nun folgende Schaltung:



Gelingt es dir, mit Hilfe der zwei Messpins nicht mehr zwei, sondern drei verschiedenen Helligkeitsstufen zu messen?

- Was misst man an `messPin1` bzw. `messPin2`, wenn der LDR stark abgedunkelt wird?
- Was misst man an `messPin1` bzw. `messPin2`, wenn der LDR etwas abgedunkelt wird?
- Was misst man an `messPin1` bzw. `messPin2`, wenn der LDR nicht abgedunkelt wird?

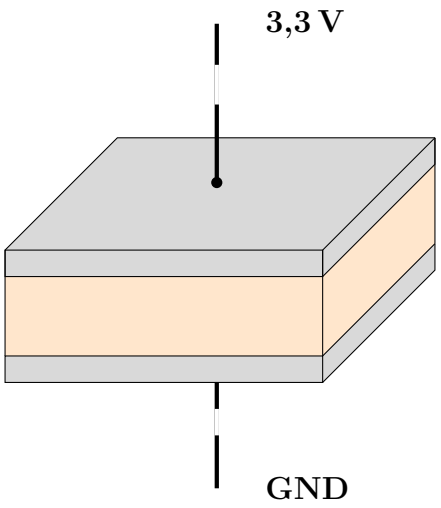
Stell das Potentiometer so ein, dass du für obige drei Fälle unterschiedliche Belegungen für `messPin1` und `messPin2` erhältst!

Helligkeit detektieren

Ergänze obige Schaltung durch einen zusätzlichen Schaltkreis mit einer LED und einem Widerstand. Schreibe ein Programm, dass die LED ganz schnell blinken lassen soll, wenn der LDR nicht abgedunkelt wird, etwas langsamer blinken lassen soll, wenn der LDR etwas abgedunkelt wird und ganz langsam blinken soll, wenn der LDR stark abgedunkelt wird!
Versichere dich vorher mit dem Voltmeter, dass du auch wirklich die drei Helligkeitsstufen unterscheiden kannst! Gegebenenfalls musst du noch die Einstellung am Potentiometer etwas verändern.

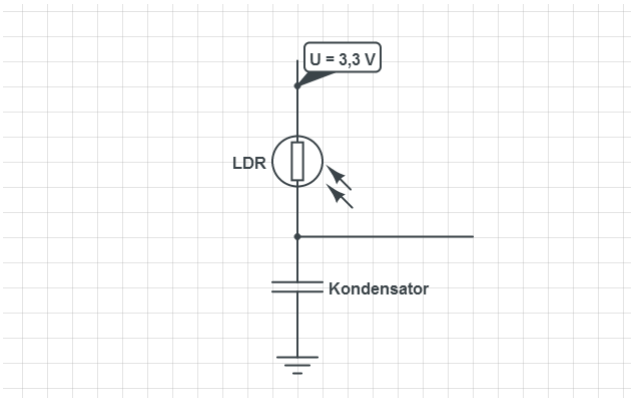
Kondensator

Ein Kondensator besteht aus zwei elektrisch leitfähigen Flächen, die durch einen Isolator getrennt sind:



Ein Kondensator kann Ladung speichern [5].

Betrachten wir folgenden Schaltplan:



Wie lange dauert es, bis „der Kondensator `HIGH` ist“? Die gemessene Zeit ist ein grobes Maß für die Dunkelheit im Raum!

Mit folgendem Programm kann man die Helligkeit im Raum unter Verwendung eines Kondensators detektieren:

```
1 from RPi.GPIO import *
2 import time
3
4 setmode(BCM)
5
6 messPin = 18
7 ledPin = 17
```

[5] Aus Elektronik für Ingenieure und Naturwissenschaftler von Hering, Bressler, Gutekunst: Die Kapazität C als Maß für das Speichervermögen des Kondensators gibt an, wie viel Ladung Q pro Spannungseinheit U gespeichert werden kann ($C = Q/U$). Die Einheit der Kapazität ist das *Farad* F: $1F = 1\,As/V$. Das heißt, ein Kondensator besitzt die Kapazität C von $1\,F$, wenn bei einem Strom von $1\,A$ innerhalb von 1 Sekunde die Spannung U auf $1\,V$ ansteigt.

```
8  setup(ledPin, OUT)
9  blinken = PWM(ledPin, 1)
10 blinken.start(50)
11
12 try:
13     while True:
14         zeit = 0 # Groesse fuer die Zeit
15         # Kondensator via messPin entladen
16         setup(messPin, OUT)
17         output(messPin, LOW)
18         # Messung durchfuehren
19         setup(messPin, IN)
20         while input(messPin) == LOW:
21             zeit = zeit+1
22
23         # Messung ausgeben - LED blinke schneller, wenn es dunkel ist
24         print(zeit)
25         blinken.ChangeFrequency(float(zeit)/40)
26 finally:
27     blinken.stop()
28     cleanup()
```

Teste obiges Programm, indem du zunächst eine geeignete Schaltung erstellst. Beachte, dass es beim Kondensator, wie bei LED, entscheidend ist, wie du ihn in das Breadboard steckst. Der kurze Fuß (weiße Strich) muss in Richtung `GND` zeigen!

Je nach Helligkeit in deinem Raum, musst du ggf. die verwendeten Parameter leicht verändern.

4.3.5 Motoren

In diesem Abschnitt lernst du verschiedene Typen von Motoren kennen und wirst mit diesen experimentieren. Wir behandeln dabei Servomotoren, DC-Motoren und Schrittmotoren.

4.3.5.1 Servomotoren

Als ersten Motoren-Typ betrachten wir einen Modellbau-Servo.

Dieser besteht aus

- einem Gleichstrommotor
- einem Getriebe
- einem Potentiometer
- und einer Steuerelektronik

Der Steuerelektronik kann über eine Signalleitung ein Stell-Wert übermittelt werden. Die Steuerelektronik dreht den Motor dann so lange in die richtige Richtung, bis sie mit Hilfe des Potentiometers den Stell-Wert misst.

Aufbau

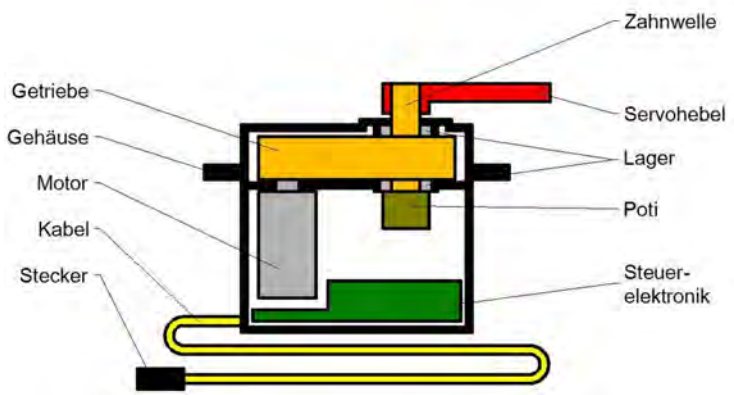
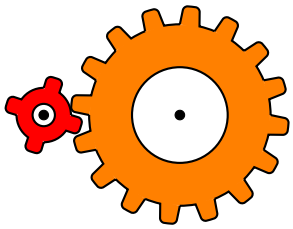


Abbildung 4.7: Aufbau eines Modellbau-Servos

Getriebe

Wird ein kleines Zahnrad (rot) verwendet, um ein großes Zahnrad (orange) anzutreiben, dann dreht sich das große Zahnrad langsamer:

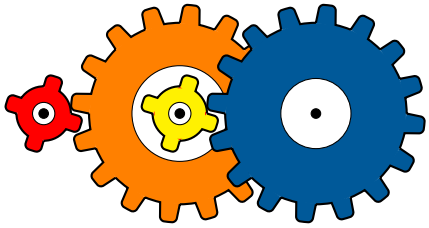


Beispiel 1:

Kleines Zahnrad mit 4 Zähnen und großes Zahnrad mit 16 Zähnen. Wie oft muss sich das kleine Zahnrad drehen, damit sich das große Zahnrad um 360° dreht?

Beispiel 2 - größere Übersetzung:

Das orange Zahnrad ist über eine Achse fest mit dem kleinen gelben Zahnrad verbunden. Die zwei kleinen Zahnräder haben je 4 Zähne, die beiden großen Zahnräder je 16 Zähne. Wie oft muss das rote Zahnrad gedreht werden, damit sich das blaue Zahnrad um 360° dreht?



Regelkreis

Der Steuerelektronik kann über eine Signalleitung ein Stell-Wert übermittelt werden.
Die Steuerelektronik dreht den Motor dann so lange in die richtige Richtung, bis sie mit Hilfe des Potentiometers (siehe Abbildung 4.7) den Stell-Wert misst.

Ziel:

Wir möchten nun den Schwenkarm eines Servomotors softwareseitig auf einen Winkel zwischen 0° und 170° (z.B. in Schritten von einem Grad) einstellen. Dabei haben wir aufgrund der digitalen Ausgänge des Raspberry Pis folgendes Problem:

- Die Ausgänge des Pis können nur mit `HIGH` oder `LOW` beschaltet werden - wir können der Steuerelektronik also nur zwei Werte übermitteln

Eine mögliche Lösung dieses Problems hast du bereits kennengelernt: **Pulsweitenmodulation**!

- Einen Ausgang im schnellen Wechsel zwischen `HIGH` und `LOW` beschalten
- Ein Mittelwert kann gemessen werden, der umso kleiner ausfällt, je länger eine `LOW`-Phase dauert

Hardware-Pulsweitenmodulation

Bisher haben wir immer die Software-Pulsweitenmodulation verwendet. Für das Ansteuern des Servomotors ist die Software-PWM allerdings ungeeignet, da sie zu langsam und ungenau ist. Der Raspberry Pi bietet **nur auf Pin 18** noch eine andere Möglichkeit für PWM an: die Hardware-PWM. Dazu benötigen wir die Bibliothek `wiringpi2`.

Die Hardware-PWM ist etwas umständlich und unterscheidet sich in den Befehlen von der Software-PWM. Die einzelnen Befehle stellen wir dir zunächst vor:

- „Tick-Konfiguration“ der Hardware-PWM
  - die Hardware-PWM stellt 19 200 000 „Ticks“ pro Sekunde zur Verfügung
  - wir vergröbern auf $19\,200\,000/192 = 100\,000$ Ticks pro Sekunde - das reicht uns aus
  - diese Konfiguration erreicht man mit folgendem Befehl:

```
1 wpi.pwmSetClock(192)
```

- „Range-Konfiguration“ der Hardware-PWM
  - wir haben nun 100 000 Ticks pro Sekunde, also ein Tick jede $0,01\,ms$
  - die Regelelektronik von Modellbau-Servos arbeitet mit einem 50 Hz Signal auf der Signalleitung
  - 50 Hz entspricht einer Periodenlänge von $20\,ms$
  - Wegen $2000 \cdot 0,01\,ms = 20\,ms$ müssen wir weiter einstellen:

```
1 wpi.setRange(2000)
```

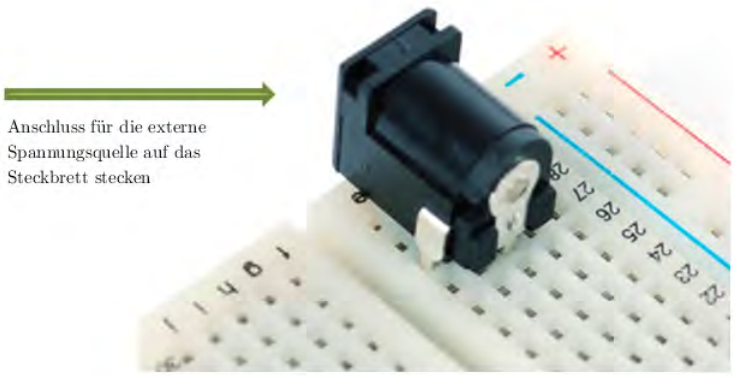
- Einstellen des DutyCycles
  - *Erinnerung:* der DutyCycle gibt an, welchen Bruchteil der Periode das Signal `HIGH` ist
  - bei unseren Servos sind Werte von ca. 50 ‰ bis ca. 220 ‰ sinnvoll ⇒ Ausprobieren!

```
1 wpi.pwmWrite(pin, 50) # Schwenkarm auf 0 Grad
2
3 wpi.pwmWrite(pin, 220) # Schwenkarm bei ca. 170 Grad
```

Externe Spannungsquelle verwenden

Wenn der Servo-Schwenkarm seine Richtung wechseln soll, dann muss die Regelelektronik intern die Spannung umpolen - das kann die Spannungsquelle „belasten“. Aus diesem Grund:

!! Servo unbedingt mit einer externen Spannung versorgen !!

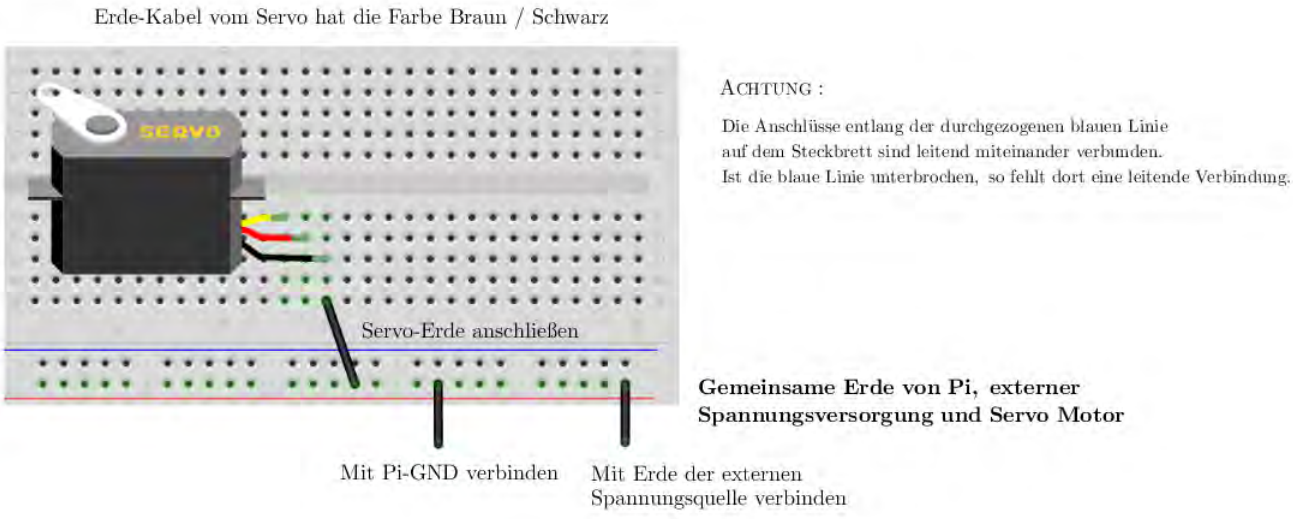


ACHTUNG :

Füße des Anschlusses NICHT leitend verbinden!
Drei **freie** Spalten auf dem Steckbrett verwenden

Anschluss auf dem Streckbrett korrekt verdrahten:

Wichtig: „Drei mal Erde“ verbinden

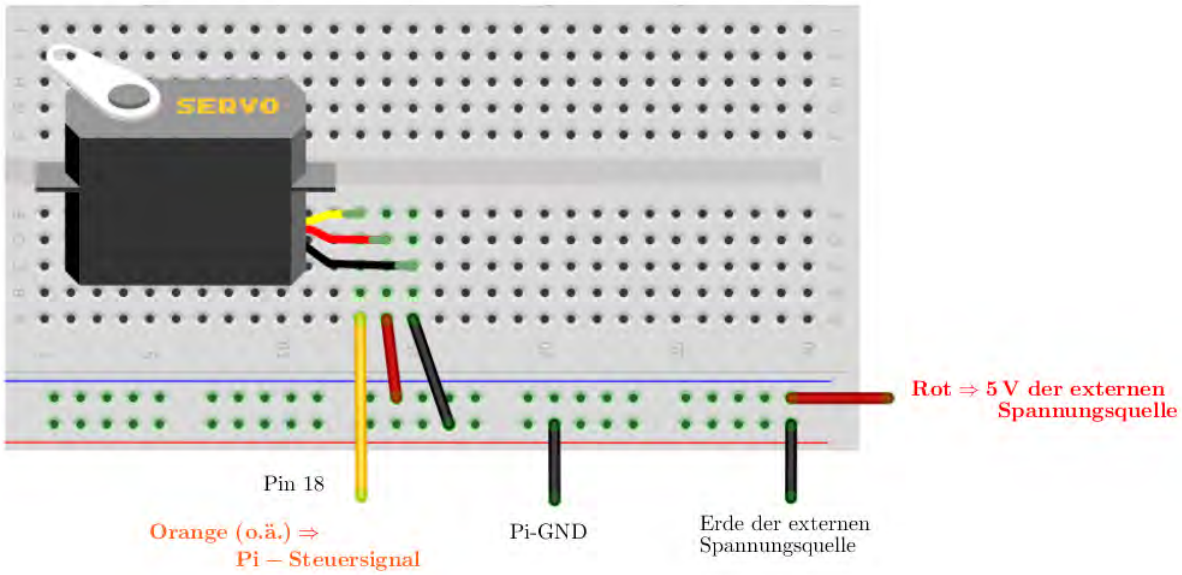


ACHTUNG :

Die Anschlüsse entlang der durchgezogenen blauen Linie
auf dem Steckbrett sind leitend miteinander verbunden.
Ist die blaue Linie unterbrochen, so fehlt dort eine leitende Verbindung.

Gemeinsame Erde von Pi, externer Spannungsversorgung und Servo Motor

5 V an rot; Signal an gelb/orange



Programm

```
1  import time
2  import wiringpi2 as wpi
3  pwm_pin = 18
4
5  # Pin und PWM Setup
6  wpi.wiringPiSetupGpio() # Pin-Nummern vom Cobbler koennen verwendet werden
7  wpi.pinMode(pwm_pin, 2) # Spannung messen: 0 - Spannung anlegen: 1 - PWM-Signal anlegen: 2
8  wpi.pwmSetMode(0) # PWM_MODE_MS --> MS steht fuer mark:space - Rechtecksignal
9  wpi.pwmSetClock(192)
10 wpi.pwmSetRange(2000)
11
12 # Starteinstellung
13 wpi.pwmWrite(pwm_pin, 50) # 50 entspricht ca. 0 Grad, 220 entspricht ca. 170 Grad
14
15 raw_input('Stellung mit Enter bestaetigen...')
16
17 # Schwenkarm bewegen
18 for i in range(3):
19     for i in range(171):
20         wpi.pwmWrite(pwm_pin, 50+i)
21         time.sleep(0.01)
22     for i in range(171):
23         wpi.pwmWrite(pwm_pin, 220-i)
24         time.sleep(0.01)
25
26 raw_input('Stellung mit Enter bestaetigen...')
27 wpi.pinMode(pwm_pin, 0)
28 wpi.pwmWrite(pwm_pin, 0)
```

Was bewirkt obiges Programm? Ausprobieren! Ändere das Programm so ab, dass sich der Schwenkarm des Servomotors 5 mal zwischen der 0°-Stellung und der 170°-Stellung hin und her bewegt.

Messwert auslesen und Motor einstellen

Wir möchten nun in einer Schaltung einen Messwert auslesen und je nachdem ob wir `HIGH` oder `LOW` messen, den Schwenkarm auf 170° oder 0° einstellen.
Für die „Messschaltung“ kannst du z.B. wieder einen LDR und ein Potentiometer verwenden und das Potentiometer so einstellen, dass an ihm als Spannungsabfall `HIGH` gemessen wird, falls der LDR nicht abgedunkelt wird und `LOW` gemessen wird, falls der LDR abgedunkelt wird.
Dann benötigst du noch eine „Servoschaltung“, um die Position des Schwenkarms ändern zu können - denke daran, dass die Hardware-PWM nur an Pin 18 möglich ist!

Hinweis: Vergiss nicht, die benötigten Module (z.B. `wiringpi2`) zu importieren.

Folgende Befehle können dir bei der Umsetzung der Aufgabenstellung helfen:

```
1 wpi.pinMode(mess_pin, 0) # Spannung messen - entspricht setup(mess_pin, IN)
2
3 m = wpi.digitalRead(mess_pin) # in m speichern, ob am Mess-Pin HIGH oder LOW
      gemessen wird
4
5 if m == 1: # 1 steht hier fuer HIGH
6     wpi.pwmWrite(pwm_pin, 220)
```

Spannung anlegen

Du hast nun bereits Befehle kennen gelernt, mit denen du die Hardware-PWM an Pin 18 steuern bzw. mit Pins Spannung messen kannst. Mit folgenden Befehlen kannst du an Pins Spannung anlegen bzw. nicht anlegen:

```
1 wpi.pinMode(anderer_pin, 1) # U anlegen - enstpricht setup(anderer_pin, OUT)
2 wpi.digitalWrite(anderer_pin, 0) # U = 0 Volt
3 wpi.digitalWrite(anderer_pin, 1) # U = 3,3 Volt
```

Projektaufgabe: Waving Flag

Mit Hilfe eines Servos kann man z.B. eine Fahne schwenken. Hierzu gibt es unter

http://raspi.tv/2013/how-to-make-your-own-raspberry-pi-flag-waving-demo

eine schöne Anleitung!

4.3.5.2 Exkurs: IC (Integrated Circuit) ULN2803a

In den folgenden zwei Abschnitten stellen wir zwei weitere Arten von Motoren vor, zuerst DC-Motoren und abschließend noch Schrittmotoren. Um diese verwenden zu können, benötigen wir ein weiteres Schaltelement - einen `ULN2803a`.
In diesem Abschnitt erklären wir dir, wie ein `ULN2803a` funktioniert und warum wir diesen benötigen.

Probleme bei Motoren:

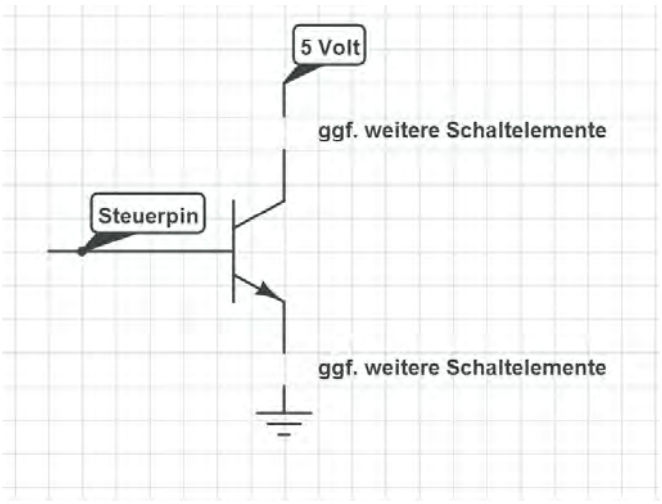
- **DC-Motoren**

  Es fließt ein Strom durch den Motor, der für die programmierbaren Ausgänge des Raspberry Pi zu groß ist.
  In DC-Motoren ist eine Spule verbaut. Wenn man eine Spule „ausschaltet“, wird ein elektrisches Feld induziert. Eine so entstehende Spannungsspitze könnte den Raspberry Pi schädigen.

- **Schrittmotoren**

  Die Schrittmotoren, die wir verwenden, benötigen eine 5 Volt Spannungsquelle. Die programmierbaren Ausgänge des Raspberry Pi liefern allerdings nur 3,3 Volt - es gibt nur einen Pin, der konstant 5 Volt Spannung liefert. Wir möchten via Programm an den Eingängen des Schrittmotors dennoch gerne 5 Volt Spannung anlegen oder eben nicht anlegen können.

Greifen wir zunächst die Problematik bei Schrittmotoren auf. Wir möchten die konstante 5 Volt Spannung an einzelnen Pins an- bzw. abschalten können. Betrachten wir folgendes Szenario:

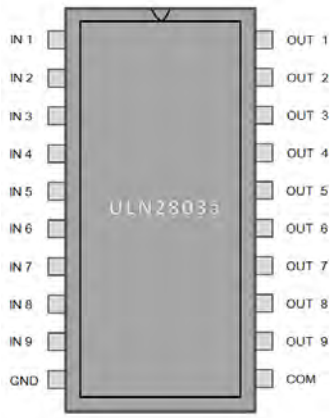


Der Stromkreis soll geschlossen sein, wenn am Steuerpin `HIGH` anliegt und nicht geschlossen sein, wenn am Steuerpin `LOW` anliegt, d.h. das mittlere Schaltelement fungiert als Schalter, der geschlossen ist, wenn an Steuerpin `HIGH` anliegt und ansonsten offen ist.

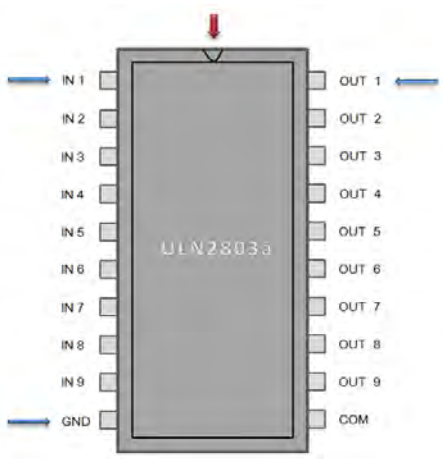
Das Schaltelement in der Mitte nennt man **Transistor**.

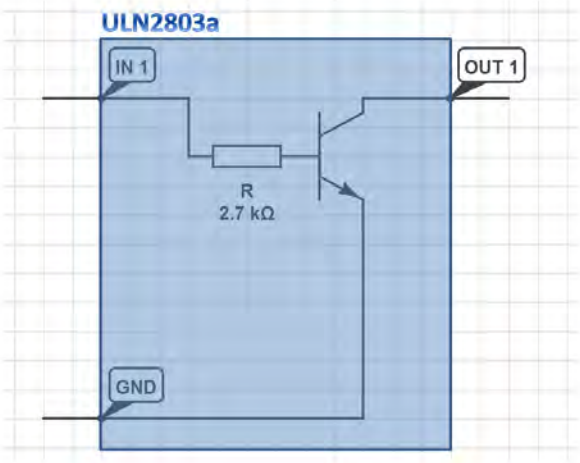
Mit Hilfe eines Tranistors können wir mit den programmierbaren Pins des Raspberry Pi nun Schaltkreise mit 5 Volt Spannung schließen oder „öffnen“.

In dem Schaltelement `ULN2803a` sind neun Transistoren verbaut. Der prinzipielle Aufbau ist auf folgendem Bild zu sehen:



Es sind viele Ein- (`IN`) und Ausgänge (`OUT`) vorhanden. Wir benötigen später vier zusammengehörende Eingans-/Ausgangspaare, Erde (`GND`) und einen Anschluss, der Transistoren vor Spannungsspitzen schützt (`COM`). Betrachten wir den internen Aufbau des `ULN2803a` anhand eines Eingans-/Ausgangspaares genauer:





Der Anschluss `GND` des `ULN2803a` muss mit `GND` des Raspberry Pis verbunden werden!

In unserer leicht vereinfachten Sichtweise können wir vorerst festhalten:

- liegt an Eingang `IN 1` `HIGH` an, wird Ausgang `OUT 1` auf Erde „durchgeschalten“

Im Abschnitt zu DC-Motoren werden wir uns noch genauer mit dem Aufbau und Funktionsweise des `ULN2803a` beschäftigen. Für den Abschnitt zu Schrittmotoren reicht das bisherige Wissen aus!

4.3.5.3 Schrittmotoren

Als letzten Motorentyp nehmen wir noch den Schrittmotor genauer unter die Lupe:

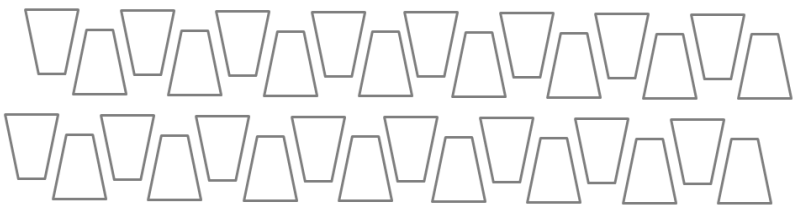
Schrittmotoren sind zwischen DC-Motoren und Servomotoren einzuordnen. Sie lassen sich auf der einen Seite exakt einstellen, einen „Schritt“ nach vorne oder hinten bewegen, auf der anderen Seite aber auch permanent rotieren.

Aufbau

Zunächst befassen wir uns mit dem Aufbau eines Schrittmotors. Dazu werfen wir einen Blick in das Innere des Motors:

Das linke Bild ist nach Öffnen des Deckels des Schrittmotors entstanden. Das Zahnrad, an dem der „Arm“ des Schrittmotors befestigt ist benötigt 32 Schritte pro Umlauf. Insgesamt hat man von diesem Zahnrad zum kleinen Zahnrad in der Mitte, das oben an der Trommel im rechten Bild befestigt ist, eine Übersetzung von 1 zu 64, d.h. das kleine Zahnrad in der Mitte muss $32 \cdot 64 = 2048$ Schritte machen, damit sich der „Arm“ des Schrittmotors einmal um 360° dreht.

Auf dem rechten Bild sieht man zwei zueinander leicht versetzte Reihen ineinander verzahnter „Zähne“. Jede Reihe besteht aus 16 Zähnen, davon 8 oben und 8 unten. Um anschließend die Funktionsweise des Schrittmotors besser erklären zu können, verwenden wir folgendes Modell, in dem die beiden Reihen dargestellt sind:



Doch wie dreht sich die Trommel? Wie wird vorwärts- und rückwärtsdrehen realisiert? Diese fragen klären wir als nächstes.

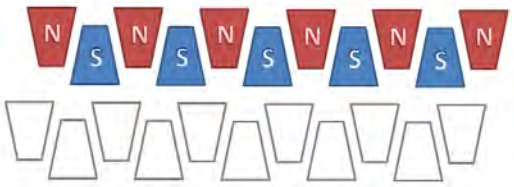
Funktionsweise

Die Trommel des Schrittmotors besteht aus einzelnen Magnetstreifen, abwechselnd Nord- und Südpol. Dabei handelt es sich um Dauermagneten[a]. Dies kann man anhand eines magnetischen Schraubenziehers leicht nachprüfen - siehe dazu das mittlere Bild auf der vorherigen Seite.
Insgesamt befinden sich acht Nordpole und acht Südpole auf der Trommel:

[a]Dauermagneten haben und halten ein statisches Magnetfeld, ohne dass ein elektrischer Stromfluss benötigt wird. Dauermagneten besitzen an ihrer Oberfläche je einen oder mehrere Nord- und Südpole (vgl. http://de.wikipedia.org/wiki/Dauermagnet).

Um die zwei Reihen ineinander verzahnter Zähne befindet sich je eine Spule (linkes Bild).[6] Wir zeigen die Funktionsweise der Spulen anhand der oberen Reihen - beachte, dass die oberen Zähne der oberen Reihe auf nachfolgenden Bildern nicht zu sehen ist!

Bewegt sich der Strom der Spule gegen den Uhrzeigersinn (mittleres Bild), wird die obere Reihe so elektromagnetisch aufgeladen, dass bei den oberen Zähnen ein Nordpol und bei den unteren Zähnen ein Südpol ist:



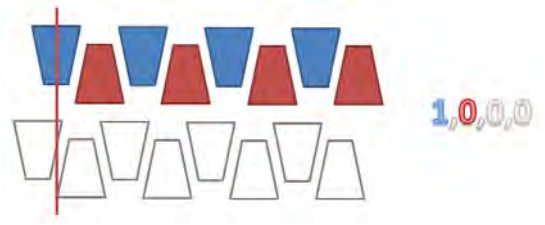
Fließt der Strom der Spule im Uhrzeigersinn (rechtes Bild), wird die obere Reihe so elektromagnetisch aufgeladen, dass bei den oberen Zähnen ein Südpol und bei den unteren Zähnen ein Nordpol ist:

Ein Schrittmotor hat fünf Anschlüsse, wobei ein Anschluss der Spannungsversorgung dient (5 Volt). Mithilfe der anderen vier Anschlüsse kann man die beiden Reihen elektromagnetisch aufladen und somit die Trommel mit den Dauermagneten zum Drehen bringen. Wie das funktioniert, betrachten wir nun in unserem Modell.

[6]Diese beiden Spulen sind völlig unabhängig voneinander

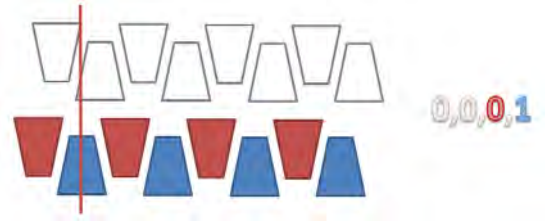
Wir betrachten einen Nordpol-Magnetstreifen (roter Strich) der Trommel und zeigen, wie dieser wandert (und sich folglich die Trommel dreht):

- Initial laden wir die obere Reihe derart elektromagnetisch auf, dass sich bei den oberen Zähnen Südpole und bei den unteren Zähnen Nordpole befinden. Dies erreichen wir, indem wir beim ersten Anschluss `HIGH` (in diesem Fall durch die Zahl 1 repräsentiert) und bei den anderen drei Anschlüssen `LOW` (in diesem Fall durch die Zahl 0 repräsentiert) anlegen:



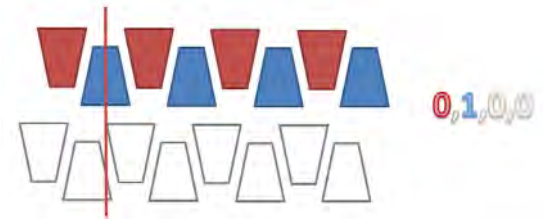
  Der von uns näher betrachtete Nordpol-Magnetstreifen der Trommel bewegt sich zu einem „Südpol-Zahn“ der oberen Reihe.

- Wir wollen nun den Nordpol-Magnetstreifen weiter bewegen. Dazu stoppen wir den elektrischen Stromfluss in der oberen Reihe und laden die untere Reihe derart elektromagnetisch auf, dass sich bei den oberen Zähnen Nordpole und bei den unteren Zähnen Südpole befinden:



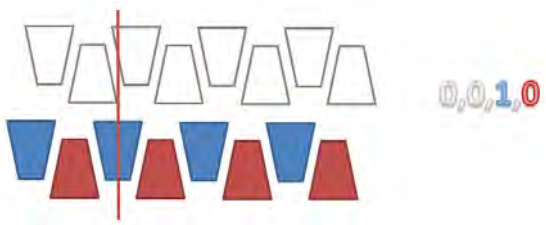
  Unser Nordpol-Magnetstreifen wandert zum ihm nächst gelegenen Südpol.

- Als nächstes stoppen wir den elektrischen Stromfluss in der unteren Reihe und laden wieder die obere Reihe elektromagnetisch auf, allerdings sollen sich nun bei den oberen Zähnen Nordpole und bei den unteren Zähnen Südpole befinden:



  Unser Nordpol-Magnetstreifen wandert zum ihm nächst gelegenen Südpol.

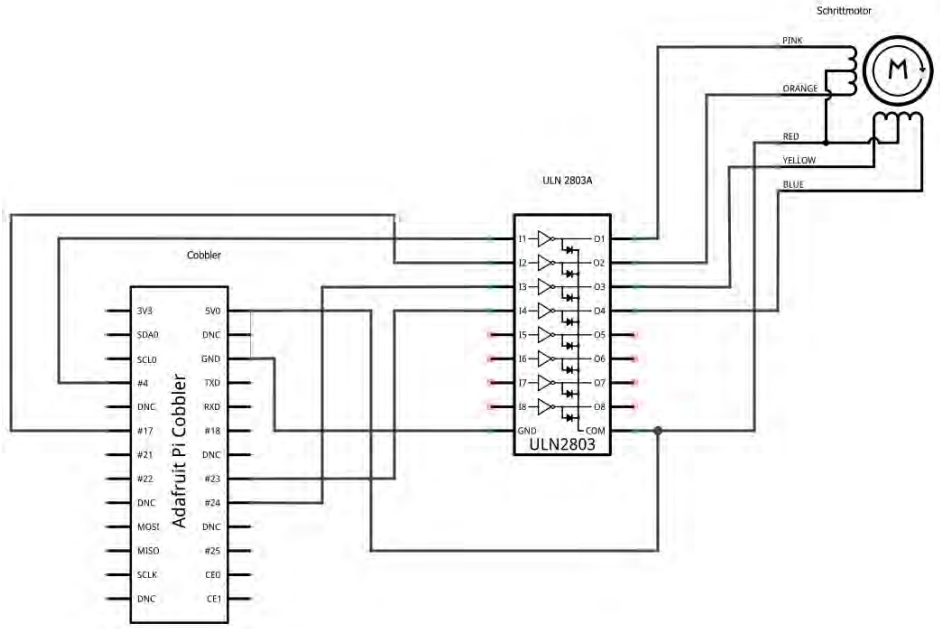
- Nun stoppen wir den elektrischen Stromfluss in der oberen Reihe und laden erneut die untere Reihe elektromagnetisch auf, allderings sollen sich nun bei den oberen Zähnen Südpole und bei den unteren Zähnen Nordpole befinden:



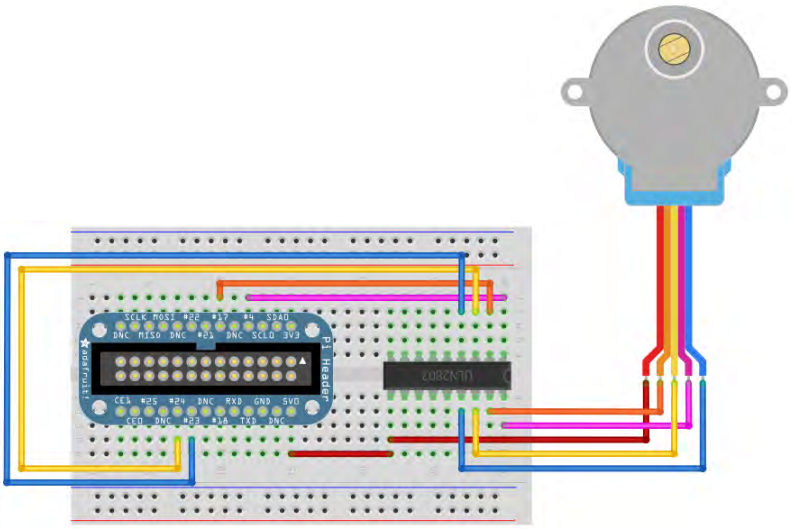
Nun geht es wieder mit der ersten Konfiguration weiter, usw. Jede Reihe besitzt je 16 Zähne, d.h. unser Nordpol-Magnetstreifen muss insgesamt 32 Schritte machen, damit er sich einmal um 360° gedreht hat. Somit muss obiger Vier-Schritt-Zyklus achtmal durchlaufen werden, damit die Trommel und somit auch das kleine Zahnrad in der Mitte eine ganze Drehung macht.
Berücksichtigt man die Übersetzung von 1 zu 64, muss der Vier-Schritt-Zyklus also $8 \cdot 64 = 512$ mal durchlaufen werden, damit sich der „Arm“ des Schrittmotors um 360° dreht.

Schaltplan und seine Umsetzung

Der Vollständigkeit halber geben wir den Schaltplan für den Schrittmotor an:



Da der Schaltplan mit dem Darlington Array `ULN2803a` und dem Schrittmotor schwer nachzuvollziehen ist, geben wir in diesem Fall zum besseren Verständnis noch die Umsetzung des Schaltplans an:



Die Funktionsweise des `ULN2803a` haben wir bereits bei den DC-Motoren ausführlich erläutert.
Der Schrittmotor hat fünf Anschlüsse - der rote Anschlüsse dient der 5 Volt-Spannungsversorgung, mit den anderen vier Anschlüssen (Steuer-Pins) können wir, wie bereits angesprochen, die Spulen elektromagnetisch aufladen.

Das Programm

Abschließend stellen wir noch ein mögliches Programm vor, mit dem man einen Schrittmotor ansteuern kann.

Zunächst müssen wir vier Pins des Raspberry Pi als Ausgänge festlegen, mit denen wir an den Steuer-Pins über den `ULN2803a` 5 Volt Spannung versorgen oder nicht anlegen können:

```
1  import RPi.GPIO as GPIO
2  import time
3
4  GPIO.setmode(GPIO.BCM)
5
6  # Variablen fuer die vier Pins
7  spule_A_1 = 4
8  spule_A_2 = 17
9  spule_B_1 = 23
10 spule_B_2 = 24
11
12 # Festlegen der vier Pins als Ausgaenge
13 GPIO.setup(spule_A_1,GPIO.OUT)
14 GPIO.setup(spule_A_2,GPIO.OUT)
15 GPIO.setup(spule_B_1,GPIO.OUT)
16 GPIO.setup(spule_B_2,GPIO.OUT)
```

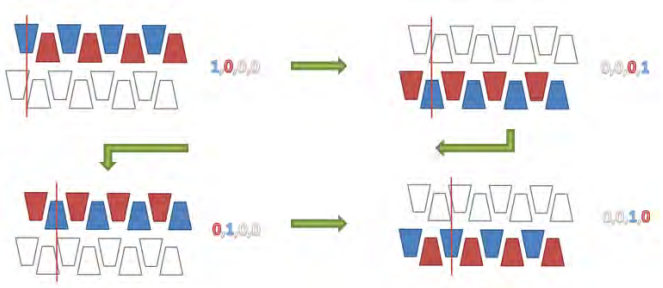
Als nächstes schreiben wir uns eine Hilfsmethode, mit der wir an den vier Pins entweder Spannung anlegen oder nicht anlegen können:

```
1  def setStep(w1,w2,w3,w4):
2      GPIO.output(spule_A_1,w1)
3      GPIO.output(spule_A_2,w1)
```

```
4       GPIO.output(spule_B_1,w1)
5       GPIO.output(spule_B_2,w1)
```

Im Abschnitt zur Funktionsweise des Schrittmotors haben wir gesehen, dass die Schritt-Sequenz in Vier-Schritt-Zyklen ablaufen. Aus diesem Grund macht es Sinn, eine Methode `forward(delay,steps)` zu schreiben, die einen Vier-Schritt-Zyklus `steps`-mal wiederholt. Der Parameter `delay` wird als Verzögerung zwischen den einzelnen Pin-Konfigurationen benötigt, da ansonsten die Umpolungen zu schnell ablaufen und sich der Schrittmotor sich somit nicht bewegt - als sinnvoller Wert für `delay` schlagen wir 5 Millisekunden vor.

```
1  def forward(delay,steps):
2      for i in range(steps):
3          setStep(1,0,0,0)
4          time.sleep(delay)
5          setStep(0,0,0,1)
6          time.sleep(delay)
7          setStep(0,1,0,0)
8          time.sleep(delay)
9          setStep(0,0,1,0)
10         time.sleep(delay)
```



Die Werte für `delay` und `steps` können wir nun den Benutzer interaktiv eingeben lassen:

```
1  try:
2      while True:
3          delay = raw_input("Wartezeit zwischen Schritten (in Millisekunden)")
4          steps = raw_input("Wie viele Schritte vorwaerts?")
5          forward(int(delay)/1000.0,int(steps)
6  finally:
7      cleanup()
```

Experimentiere mit dem Programm!

Aufgaben

(a) Welcher Wert für `steps` ergibt eine ganze Umdrehung?

Hinweis: Überlege dir dazu, wie oft der Vier-Schritt-Zyklus ausgeführt werden muss, dass sich der Arm des Schrittmotors um 360° dreht.

(b) Schreibe eine Methode `halfRotation()`, die den Arm des Schrittmotors um 180° dreht und teste diese.

(c) Schreibe eine Methode `backward(delay,steps)`, die den Zeiger des Schrittmotors im Uhrzeigersinn dreht!

(d) Bei dieser Aufgabe geht es darum, den Vier-Schritt-Zyklus noch zu verfeinern!
Überlege dir eine Sequenz von acht Pin-Konfigurationen, mit denen du die vier Schritte des Zyklus noch verfeinern kannst!

Hinweis: Es ist auch möglich, an beiden Spulen gleichzeitig Strom anzulegen und somit in beiden Reihen ein Magnetfeld zu induzieren!

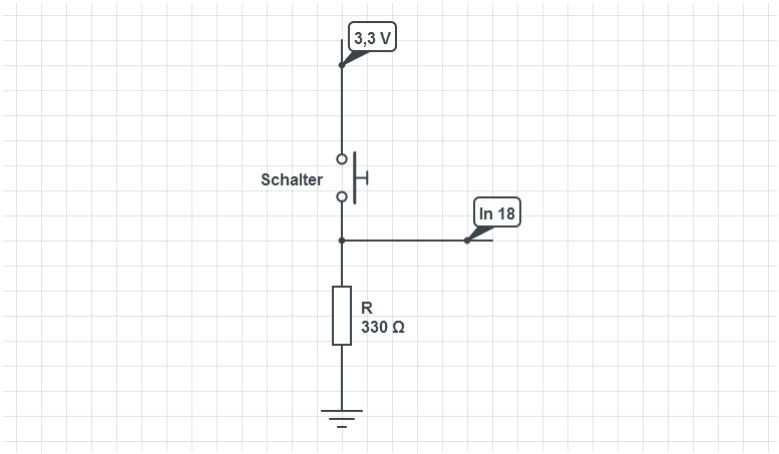
4.3.6 Minecraft

Auf dem Raspberry Pi läuft eine eingeschränkte Version des Spiels Minecraft - `Minecraft: Pi Edition`. Im Gegensatz zu der Original Version hat sie weniger Bausteine und Features, kann dafür aber z.B. über ein Python-Programm gesteuert werden. Dies wollen wir nutzen.

Im Rahmen dieser Modul-Version bieten wir hier eine mögliche Aufgabe zu Minecraft. Dabei soll aufgezeigt werden, welche Schritte nötig sind, damit das Python-Programm und Minecraft kommunizieren können.
Der Phantasie der Schüler sind dann keine Grenzen mehr gesetzt, welche Ziele sie sich stecken. In Abschnitt 4.5.2 ist noch eine mögliche Idee als Projektaufgabe angeführt.

Aufgabe: Turm bauen

Wir verwenden folgende einfache Schaltung:



Wenn wir den Schalter drücken, soll die Spielfigur um ein Pixel nach oben gesetzt und unter dieser ein Stein[7] platziert werden. Durch mehrmaliges Drücken des Schalters wird der „Turm“, auf dem die Spielfigur immer größer.

Damit das Programm mit Python kommunizieren kann, muss die Python-API eingebunden werden. Dazu muss das Python-Programm im Ordern

`/home/pi/mcpi/api/python`

abgespeichert werden!!!

[7] Und zwar ein Stein mit der gleichen ID, wie der Stein hat, auf der die Spielfigur bisher stand

Wurde das Programm im oben angegebenen Ordner gespeichert, bindet man die Minecraft-Module mit folgenden Befehlen ein:

```
1 import mcpi.minecraft as minecraft
2 import mcpi.block as block
```

Bevor du das Python-Programm startest, musst du Minecraft öffnen[8] und ein neues Spiel starten. Via `Esc` pausiert du das Minecraft Spiel. Dein Programmrumpf könnte wie folgt aussehen:

```
1  import mcpi.minecraft as minecraft
2  import mcpi.block as block
3  from RPi.GPIO import *
4
5  setmode(BCM)
6  messPin = 18
7  setup(messPin, IN)
8
9  # Verbindung zum Minecraft-Server aufbauen. Die Verbindung speichern wir in einer Variablen
       mc. Ueber diese Verbindung muessen alle Kommandos geschickt werden
10 mc = minecraft.Minecraft.create()
```

Hinweise:

- die aktuelle Spielerposition kannst du mit

```
1 mc.player.getTilePos()
```

  abfragen. Wir bekommen ein `vec3`-Objekt zurück, welches x-, y- und z-Koordinate speichert.

- mit dem Befehl

```
1 mc.player.setTilePos(x,y,z)
```

  kannst du die neue Spielerposition setzen.

- die y-Koordinate beschreibt die vertikale Achse. Wollen wir den Stein unter uns platzieren, musst du also von der y-Koordinate 1 abziehen:

```
1 posBelowPlayer = mc.player.getTilePos()
2 posBelowPlayer.y = posBelowPlayer.y - 1
```

- mit dem Befehl

```
1 mc.getBlock(x,y,z)
```

  bekommt man die Block-ID an der angegebenen Position - ein Luft-Block hat z.B. die ID `block.AIR`

[8] Desktop → Dateimanager → /home/pi/mcpi → Doppelklick auf `minecraft-pi` oder im `LXTerminal` den Befehl `sudo /home/pi/mcpi/minecraft-pi` eingeben

- mit dem Befehl

```
1 mc.setBlock(x,y,z,blockId)
```

  wird ein Stein mit der ID `blockId` auf die Position (x, y, z) gesetzt.

Auf der Internetseite

http://arghbox.files.wordpress.com/2013/06/table.pdf

findest du eine Übersicht über weitere Befehle.

Versuche nun die Aufgabenstellung umzusetzen!

4.4 Lösungen

4.4.1 Spannungen via Programm an- bzw. nicht anlegen

4.4.1.1 Blinken mit fester Anzahl

Lösungsbeispiel:

```
from RPi.GPIO import *
import time

ledPin = 17

setmode(BCM)
setup(ledPin, OUT)

for i in range(5):
    output(ledPin, HIGH) # Spannung anlegen
    time.sleep(2)        # 2 Sekunden warten
    output(ledPin, LOW)  # Spannung nicht anlegen
    time.sleep(2)        # 2 Sekunden warten
```

Falls wir die Zeilen mit `time.sleep(2)` löschen, blinkt die LED nicht mehr. Sofort nachdem die Spannung angelegt wurde, wird die Spannung nicht mehr angelegt wird. Dies geschieht so schnell, dass die LED nicht leuchtet.

4.4.1.2 Endlos Blinken

Lösungsbeispiel:

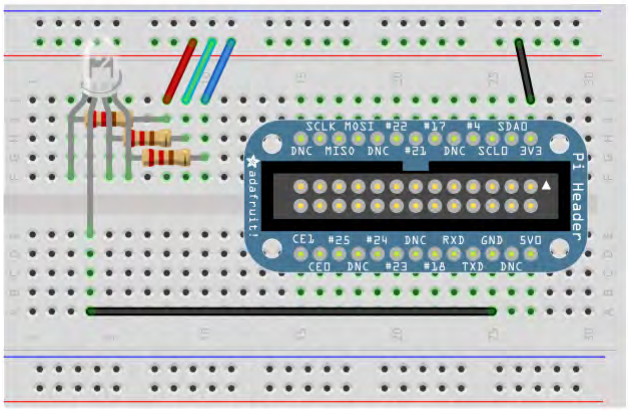
```
from RPi.GPIO import *
import time

ledPin = 17

setmode(BCM)
setup(ledPin, OUT)

try:
    while True:
        output(ledPin, HIGH) # Spannung anlegen
        time.sleep(2)        # 2 Sekunden warten
        output(ledPin, LOW)  # Spannung nicht anlegen
        time.sleep(2)        # 2 Sekunden warten
finally:
    cleanup()
```

4.4.1.3 Pulsweitenmodulation - Blinkprozess starten

Experimentieraufgabe - s. Aufgabenstellung.

4.4.1.4 PWM und RGB-LED - Blinken in Regenbogenfarben

Alle Farben zum Leuchten bringen



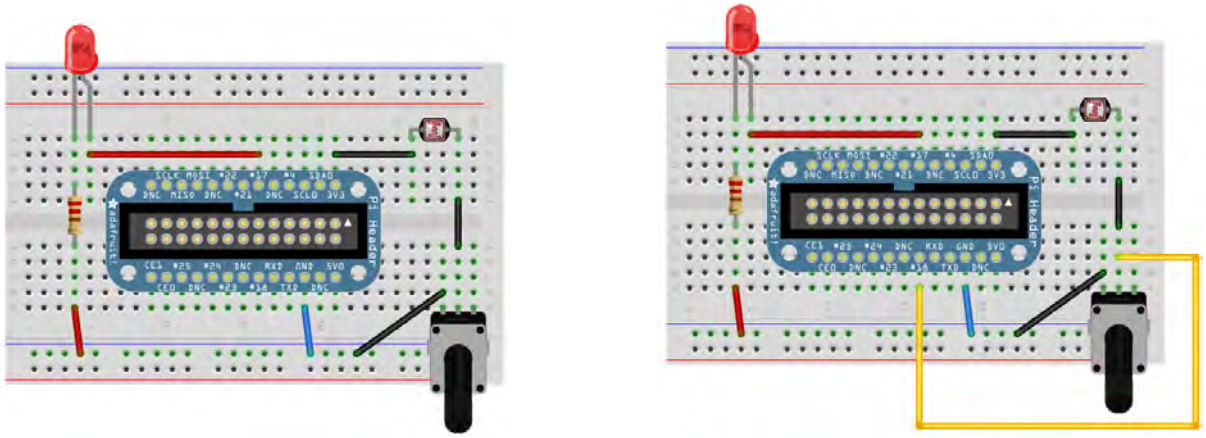
Um jede Farbe einzeln zum Leuchten zu bringen, darf man nur das rote (bzw. das grüne, bzw. das blaue) Kabel mit 3,3 V verbinden.

Pulsweitenmodulation und RGB-LED

Experimentieren - wer schafft mehr Farben?

4.4.2 Spannungen via Programm messen - LED steuern

4.4.2.1 Abdunkeln detektieren



Umsetzung des angegebenen Schaltplans (links) und Ergänzung des Schaltplans durch die „Messleitung“ (rechts).

Auslesen, ob abgedunkelt wird oder nicht

```
1 from RPi.GPIO import *
2
3 setmode(BCM)
4
5 ledPin = 17
6 messPin = 18
```

```
7
8  setup(ledPin, OUT)
9  setup(messPin, IN)
10
11 try:
12     while True:
13         # Ausgabe des an Eingang 18 gemessenen Wertes
14         print(input(messPin))
15 finally:
16     cleanup()
```

4.4.2.2 LED leuchtet, LED leuchtet nicht

```
1  from RPi.GPIO import *
2
3  setmode(BCM)
4
5  ledPin = 17
6  messPin = 18
7
8  setup(ledPin, OUT)
9  setup(messPin, IN)
10
11 try:
12     while True:
13         # wird nicht abgedunkelt...
14         if input(messPin) == HIGH:
15             # ... dann keine Spannung anlegen!
16             output(ledPin, LOW)
17         # wird abgedunkelt...
18         else:
19             # ... dann Spannung anlegen!
20             output(ledPin, HIGH)
21 finally:
22     cleanup()
```

4.4.2.3 LDR und Potentiometer vertauschen

Die LED leuchtet nun, wenn der LDR nicht abgedunkelt wird und leuchtet nicht, wenn der LDR abgedunkelt wird.

4.4.2.4 Lichtschalter

Lösung mit Einführung zweier neuer Variablen:

```
1  from RPi.GPIO import *
2  import time
3
4  setmode(BCM)
5
6  ledPin = 17
7  messPin = 18
```

```
8
9  # Variable lichtan, die speichert, ob die LED aktuell leuchtet oder nicht
10 lichtan = False
11 # Variable release, die speichert, ob der LDR zwischenzeitlich nicht mehr abgedunkelt wurde
12 release = False
13
14 setup(ledPin, OUT)
15 setup(messPin, IN)
16
17 try:
18     while True:
19         # wird der LDR nicht abgedunkelt, wird das in der Variable release gespeichert
20         if input(messPin) == HIGH:
21             release = True
22             time.sleep(0.1)
23         else:
24             # wurde zuvor einmal nicht abgedunkelt...
25             if release:
26                 # Licht an? Dann aus!
27                 if lichtan:
28                     output(ledPin, LOW)
29                     lichtan = False
30                 # Licht aus? Dann an!
31                 else:
32                     output(ledPin, HIGH)
33                     lichtan = True
34                 release = False
35                 time.sleep(0.1)
36 finally:
37     cleanup()
```

Lösung mit Haltepunkten:

```
1  from RPi.GPIO import *
2  import time
3
4  setmode(BCM)
5
6  ledPin = 17
7  messPin = 18
8
9  # Variable lichtan, die speichert, ob die LED aktuell leuchtet oder nicht
10 lichtan = False
11
12 setup(ledPin, OUT)
13 setup(messPin, IN)
14
15 try:
16     while True:
17         # Warte solange, bis der LDR abgedunkelt wird
18         while input(messPin) == HIGH:
19             pass
20         # Licht aus? Dann an! Licht an? Dann aus!
21         if lichtan:
```

```
22              output(ledPin, LOW)
23              lichtan = False
24          else:
25              output(ledPin, HIGH)
26              lichtan = True
27          time.sleep(0.1)
28          # Warte solange, bis der LDR nicht mehr abgedunkelt wird
29          while input(messPin) == LOW:
30              pass
31  finally:
32      cleanup()
```

4.4.2.5 RGB-LED durchschalten

Farben „durchschalten“

```
1   from RPi.GPIO import *
2   import time
3
4   setmode(BCM)
5
6   pinBlau = 4
7   pinGruen = 17
8   pinBlau = 22
9
10  # Methode, um Spannungen an den jeweiligen LEDs anlegen bzw. nicht anlegen zu koennen
11  def farbeAendern(w1,w2,w3):
12      output(pinBlau,w1)
13      output(pinGruen,w2)
14      output(pinRot,w3)
15
16  messPin = 18
17
18  # Variable step, die speichert, welche Farbe aktuell leuchtet
19  step = 0
20
21  setup(ledPin, OUT)
22  setup(messPin, IN)
23
24  try:
25      while True:
26          # Warte solange, bis der LDR abgedunkelt wird
27          while input(messPin) == HIGH:
28              pass
29
30          # Farben durchschalten
31          if step == 0:
32              farbeAendern(HIGH,LOW,LOW)
33              step = 1
34          elif step == 1:
35              farbeAendern(LOW,HIGH,LOW)
36              step = 2
37          else:
```

```
38                farbeAendern(LOW,LOW,HIGH)
39                step = 0
40            time.sleep(0.1)
41
42            # Warte solange, bis der LDR nicht mehr abgedunkelt wird
43            while input(messPin) == LOW:
44                pass
45    finally:
46        cleanup()
```

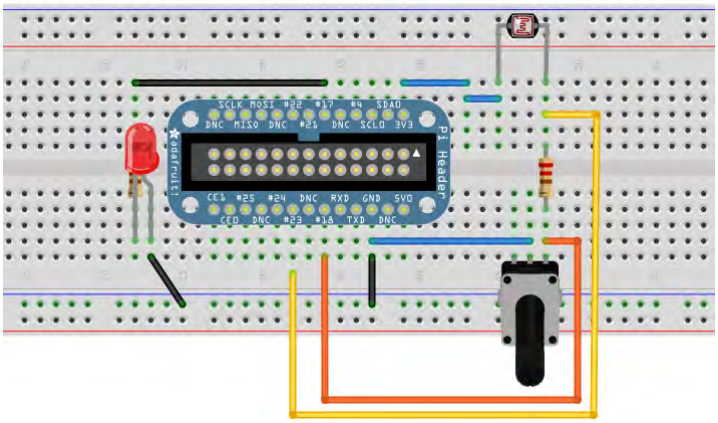
Farbwechsel

```
1   from RPi.GPIO import *
2   import time
3
4   setmode(BCM)
5
6   pinBlau = 4
7   pinGruen = 17
8   pinBlau = 22
9
10  # Methode, um Spannungen an den jeweiligen LEDs anlegen bzw. nicht anlegen zu koennen
11  def farbeAendern(w1,w2,w3):
12      output(pinBlau,w1)
13      output(pinGruen,w2)
14      output(pinRot,w3)
15
16  messPin = 18
17
18  # Variable step, die speichert, welche Farbe aktuell leuchtet
19  step = 0
20
21  setup(ledPin, OUT)
22  setup(messPin, IN)
23
24  try:
25      while True:
26          while input(messPin) == LOW:
27              # Farben durchschalten
28              if step == 0:
29                  farbeAendern(HIGH,LOW,LOW)
30                  step = 1
31              elif step == 1:
32                  farbeAendern(LOW,HIGH,LOW)
33                  step = 2
34              else:
35                  farbeAendern(LOW,LOW,HIGH)
36                  step = 0
37              time.sleep(0.1)
38  finally:
39      cleanup()
```

4.4.3 Kondensator

Mehr als hell / dunkel messen

Schaltbild:



Programm:

```
from RPi.GPIO import *
import time

setmode(BCM)

messPin1 = 18
messPin2 = 23
ledPin = 17

setup(messPin1, IN)
setup(messPin2, IN)
setup(ledPin, OUT)

blinken = PWM(ledPin, 80)
blinken.start(50)

try:
    while True:
        if input(messPin1) == HIGH and input(messPin2) == HIGH:
            blinken.changeFrequency(80)
        elif input(messPin1) == HIGH and input(messPin2) == LOW:
            blinken.changeFrequency(40)
        elif input(messPin1) == LOW and input(messPin2) == LOW:
            blinken.changeFrequency(10)
finally:
    cleanup()
```

Kondensator

Experimentierphase - siehe Aufgabenstellung.

4.4.4 Motoren

4.4.4.1 Servomotoren

4.4.4.2 DC-Motoren

4.4.4.3 Schrittmotoren

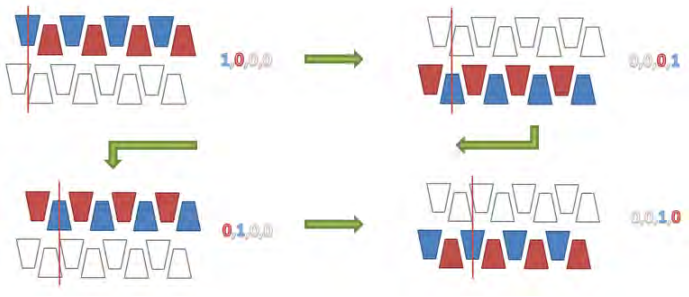
Hier findest du die Lösungen zu den Aufgaben zum Schrittmotor:

(a) Die Trommel des Schrittmotors benötigt 32 „Umpolungs"-Schritte, bis sie sich einmal um 360° gedreht hat. Somit müssen wir den Vier-Schritt-Zyklus acht mal durchlaufen.
Mit der Übersetzung von 1 zu 64, ergibt sich für `steps` folglich der Wert $8 \cdot 64 = 512$, d.h. der Aufruf der Methode `forward(5,512)` führt zu einer ganzen Drehung des Arms des Schrittmotors.

(b) In Teilaufgabe (a) haben wir berechnet, dass 512 Schritte für eine ganze Drehung benötigt werden. Folglich erreichen wir mit 256 Schritten eine halbe Drehung:

```
1  # halfRotation() unter Verwendung der Methode forward(delay,steps)
2  def halfRotation():
3      forward(5,256)
4
5  # halfRotation() ohne Verwendung der Methode forward(delay,steps)
6  def halfRotation():
7      delay = 0.005
8      for i in range(256):
9          setStep(1,0,0,0)
10         time.sleep(delay)
11         setStep(0,0,0,1)
12         time.sleep(delay)
13         setStep(0,1,0,0)
14         time.sleep(delay)
15         setStep(0,0,1,0)
16         time.sleep(delay)
```

(c) Wir kennen bereits die Vier-Schritt-Sequenz für Drehen gegen den Uhrzeigersinn (Methode `forward(delay,steps)`):

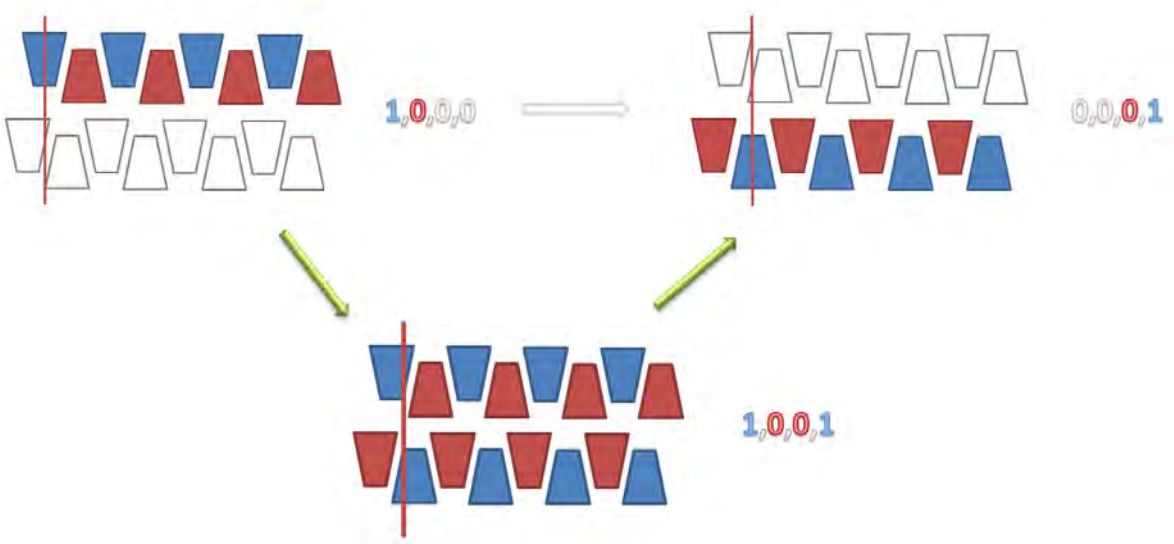


Um den Arm des Schrittmotors im Uhrzeigersinn drehen zu können, müssen wir diese Befehlssequenz invertieren, d.h. gerade in entgegengesetzter Richtung durchlaufen:

```
1  def backward(delay, steps):
2      for i in range(steps):
3          setStep(1,0,0,0)
4          time.sleep(delay)
5          setStep(0,0,1,0)
6          time.sleep(delay)
7          setStep(0,1,0,0)
8          time.sleep(delay)
9          setStep(0,0,0,1)
10         time.sleep(delay)
```

(d) Betrachten wir den ersten Schritt des Vier-Schritt-Zyklus:

Dieser Schritt lässt sich noch um einen Zwischenschritt verfeinern, und zwar, wenn wir an beiden Spulen Strom anlegen, dass wir folgende Magnetfelder induzieren:



Diese Zwischenschritte lassen sich viermal in unseren Vier-Schritt-Zyklus einfügen und wir erhalten somit einen feineren Acht-Schritt-Zyklus:

```
1  def forward(delay, steps):
2      for i in range(steps):
3          setStep(1,0,0,0)
4          time.sleep(delay)
5          setStep(1,0,0,1)
6          time.sleep(delay)
7          setStep(0,0,0,1)
8          time.sleep(delay)
9          setStep(0,1,0,1)
10         time.sleep(delay)
11         setStep(0,1,0,0)
12         time.sleep(delay)
13         setStep(0,1,1,0)
14         time.sleep(delay)
15         setStep(0,0,1,0)
16         time.sleep(delay)
17         setStep(1,0,1,0)
18         time.sleep(delay)
```

4.4.5 Minecraft

Das Python-Programm für die Aufgabe „Turm bauen“ könnte wie folgt aussehen:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
from RPi.GPIO import *
import time

setmode(BCM)
messPin = 18
setup(messPin, IN)

mc = minecraft.Minecraft.create()

try:
    while True:
        if (input(messPin) == HGIH):
            # Spieler-Position auslesen
            playerPos = mc.player.getTilePos()
            # Block-ID vom Block unter dem Spieler auslesen
            blockId = mc.getBlock(playerPos.x,playerPos.y-1,playerPos.z)
            # Spieler um ein Pixel nach oben setzen
            mc.player.setTilePos(player.x,player.y+1,player.z)
            # Block mit der eingelesen ID unter dem Spieler setzen
            mc.setBlock(playerPos.x,playerPos.y,playerPos.z,blockId)
            # Bounce-Effekt vom Button verhindern
            time.sleep(0.2)
finally:
    cleanup()
```

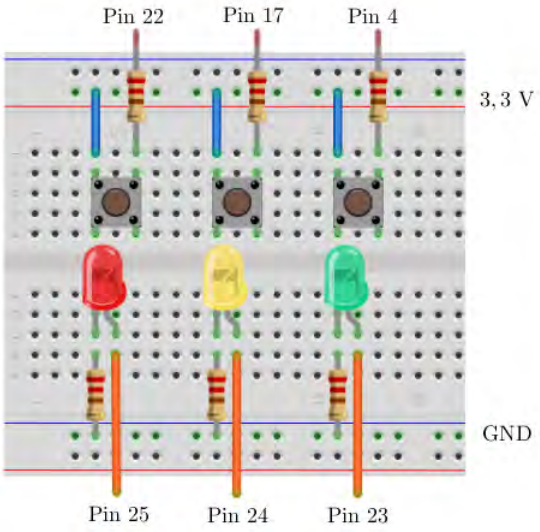
4.5 Projektaufgaben

In diesem Abschnitt stellen wir noch mögliche Ideen und Anregungen für Projektaufgaben vor, die in den Technikgruppen oder auf P-Seminaren umgesetzt werden können. Wir verzichten an dieser Stelle bewusst auf Lösungen.

4.5.1 Buttonmind

Idee: Merken der Reihenfolge blinkender LEDs und Nachdrücken mit zugehörigen Schaltern

Mögliches Schaltungsbild:



Beschreibung:
Dein Programm soll zunächst drei mal hintereinander eine der drei LEDs zum Leuchten bringen, z.B. Rot, Gelb und Grün. Der Spieler muss sich diese Reihenfolge merken und anschließend die entsprechenden Buttons drücken. In unserem Fall wäre das der linke Button, der mittlere Button und der rechte Button.
Wenn das Programm erkennt, dass der Spieler die richtige Reihenfolge gedrückt hat, soll es nun die LEDs vier mal aufblinken lassen. Die drei bisherigen LEDs bleiben gleich und es kommt noch eine LED hinzu - also z.B. Rot, Gelb, Grün, Rot. Nun wartet das Programm wieder auf die Benutzereingabe, usw.
Drückt der Spieler einen falschen Button, soll das Programm alle drei LEDs gleichzeitig einmal aufblinken lassen und sich dann beenden.

Hinweise:

- Buttons und LEDs sind physikalisch nicht miteinander verbunden
- Die interne Verknüpfung von roter LED mit rechtem Schalter kannst du z.B. sehr schön mittels Listen erreichen:

```
1 mapping = [(pin_right_button, pin_green_led), (pin_mid_button,
      pin_yellow_led), (pin_left_button, pin_red_led)]
```

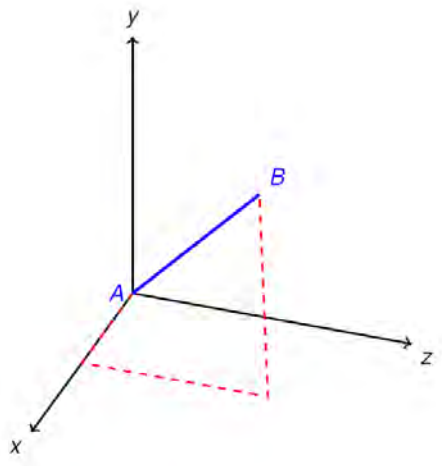
- Listen und ihre Eigenschaften werden auch in Kapitel 3 aufgegriffen.

Gehe bei der Umsetzung der Aufgabe schrittweise vor! Du kannst dich etwa zunächst nur um das Blinken der LEDs kommen und sicherstellen, dass die bisherige Reihenfolge auch beim nächsten Blinken eingehalten wird!

4.5.2 Minecraft

Idee: Es werden in einem bestimmten Umkreis vom Spieler drei Zielblöcke in die Welt gesetzt. Ziel ist es, diese zu finden und zu zerstören. Hinweise durch eine RGB-LED, die je nach Entfernung schneller oder langsamer blinkt.

Beschreibung:
Du hast bereits gesehen, wie du die Verbindung zwischen deinem Python-Programm und dem Minecraft-Spiel auf dem Raspberry Pi herstellst. Dein Programm soll zunächst die Position der Spieler-Figur auslesen und drei Blöcke in einem bestimmten Umkreis (`hiderange`) um die Spielerposition setzen. Beachte dabei, dass die Blöcke nicht irgendwo in der Luft oder unter der Erde platziert werden. Minecraft verwendet dabei folgendes Koordinatensystem:



Der größtmögliche Wert für y ist dabei 150. Ein Luftblock hat die `id block.AIR`.
Implementiere das Programm zunächst nur für einen blauen Stein (`block.LAPIS_LAZULI_BLOCK`). Je näher du dich diesem blauen Stein näherst, umso schneller soll die RGB-LED leuchten. Du könntest z.B. die Frequenz der Pulsweitenmodulation auf `hiderange/distance` setzen, wobei `distance` die aktuelle Entfernung zum Stein bezeichnet.

Wenn dein Programm für den blauen Block funktioniert könntest du, falls der blaue Stein zerstört wurde einen roten Stein (`block.MELON`) und anschließend noch einen grünen Stein (`block.GLOWSTONE_BLOCK`) setzen und die RGB-LED in der entsprechenden Farbe blinken lassen.

Hinweise:

- Schreibe zunächst eine Methode `distance(p1,p2)`, die dir den Abstand von Punkt `p1` zu Punkt `p2` berechnet. Mittels `p1.x` kannst du beispielsweise auf die x-Koordinate von Punkt `p1` zugreifen. Obige Abbildung kann dir bei deinen Überlegungen helfen!
  Quadrieren geht in Python mit `**2`; die Quadratwurzel ziehst du mit `**0.5`
- Überlege dir genau, wie du geeignete x-, y- und z-Koordinaten für den blauen Stein berechnen kannst, so dass (1) der Stein innerhalb einer gewissen Grenze `hiderange` und nicht in der Luft oder unter der Erde platziert wird.

Hinweis:

- Auf der Internetseite

  http://arghbox.wordpress.com/tag/minecraft-pi/

  sind viele Minecraft-Befehle erklärt. Auf der Internetseite

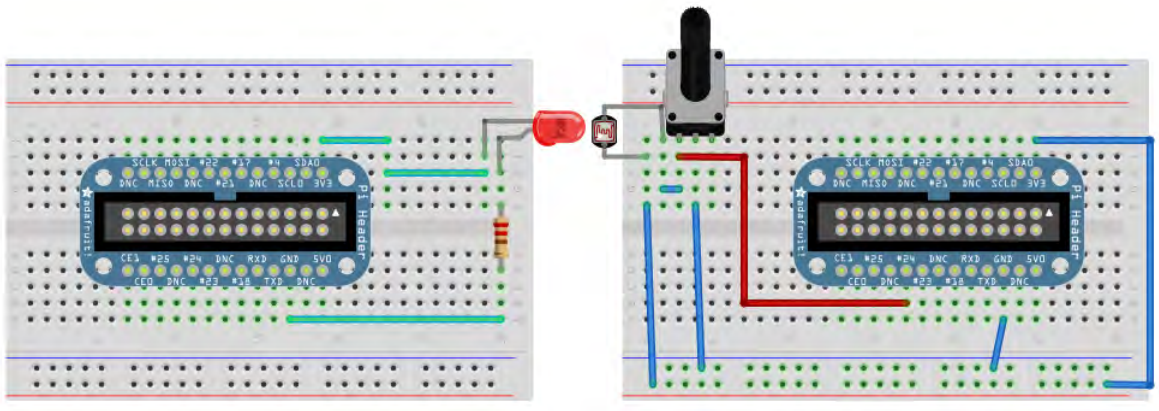
  http://arghbox.files.wordpress.com/2013/06/table.pdf

  findest du eine Übersicht über die wichtigsten Minecraft-Befehle.

Viel Spaß bei der Umsetzung dieses Minecraft-Projektes. Du kannst dir natürlich noch viele neue Aufgaben im Bezug auf Minecraft ausdenken!

4.5.3 Datenkanal (schwer)

Idee: Datenkanal zwischen zwei Raspberry Pis aufbauen, der es uns erlaubt, Text von einem Pi zum anderen zu übertragen (unidirektional).



Zur Kommunikation benötigen wir grundsätzlich zwei Dinge:

- ein Mittel (oder Medium) zur Kommunikation
- ein Protokoll, das die Regeln für die Kommunikation festlegt

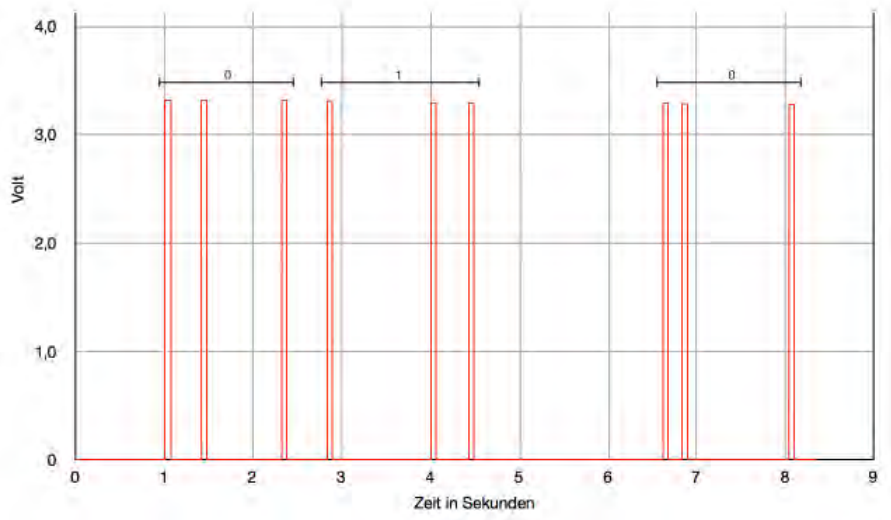
Das Medium

Als Medium verwenden wir Licht. Den eingegebenen Text wandeln wir in eine Bitfolge (Folge von Nullen und Einsen) um. Indem wir eine LED aufblinken lassen, teilen wir dem zweiten Raspberry Pi diese Bitfolge mit. Dieser detektiert das Blinken mittels eines Fotowiderstandes.

Das Protokoll

Immer wenn zwei Geräte oder Programme miteinander kommunizieren, muss man für diese Kommunikation ein Protokoll festlegen. Etwa legt das `HTTP`[9]-Protokoll fest, welche Schritte zwischen einem Internetbrowser und einem Internetserver notwendig sind, damit der Browser eine Webseite anzeigen kann.
Bei unserer Aufgabe musst du dir überlegen, wie der Empfänger-Pi feststellt, ob der Sender-Pi ihm eine 0 oder eine 1 mitteilen will.
Wir haben uns folgendes Protokoll für die Kommunikation überlegt:

- pro Bit (0 oder 1) blinkt der Sender-Pi dreimal
- das erste und das dritte Blinken legen einen äußeren Rahmen fest, in dem das mittlere Blinken positioniert wird
- der Empfänger-Pi wartet auf dreimaliges Ansteigen und Abfallen der Signalflanke vom Fotowiderstand und entscheidet dann, ob das mittlere Ansteigen/Abfallen in der ersten (0) oder in der zweiten Hälfte (1) des Rahmens lag:



[9] `HTTP` steht für Hyptertext Transfer Protokoll

Hinweise:

- um einen Buchstaben in eine Bitfolge umzuwandeln, musst du dich über den ASCII-Code informieren
- du musst das Potentiometer wieder so einstellen, dass du mit der Messleitung nach dem Fotowiderstand das Blinken der Sender-LED detektieren kannst - je nach Lichtverhältnisse kann das variieren
- sende zunächst nur einen Buchstaben
- unsere Kommunikation ist störungsanfällig. Sobald ein Bit falsch übertragen wurde, kann der Text nicht mehr korrekt angezeigt werden

Aktuelle Forschung

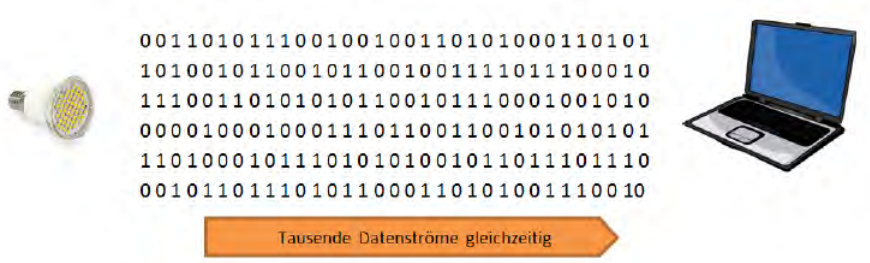
Dieses Thema scheint auch in der aktuellen Forschung hochaktuell zu sein. Forscher arbeiten daran, dass man künftig mit einer LED im Internet surfen kann. Mehr dazu kannst du unter

http://www.heute.de/Daten-aus-der-Deckenlampe-30668406.html

und

http://www.ted.com/talks/lang/de/harald_haas_wireless_data_from_every_light_bulb.html

erfahren.



Kapitel 5

Quellenangaben

In diesem Abschnitt sind die Quellen verwendeter Bilder angeführt, falls dies nicht bereits auf der jeweiligen Seite geschehen ist:

- Python Logo auf der Titelseite und den Seiten 7 und 14

  http://www.tagmotion.de/wp-content/uploads/2009/09/google_android_logo.jpg

- Schaltpläne - Alle Schaltpläne wurden mit `CircuitLab` erstellt:

  https://www.circuitlab.com/

- Schaltplanumsetzungen - Alle Umsetzungen von Schaltplänen wurden mit Fritzing erstellt:

  http://fritzing.org/home/

- Bilder im Abschnitt Hardware auf Seite 11

  https://www.modmypi.com/

- Glühbirne auf den Seiten 20, 34, 37 und 76

  http://gluehbirne.ist.org/images/gluehbirne/500px-Dialog-information_on.svg.png

- Multimeter-Messkabel rot auf den Seiten ?? und 73

  http://jto.tinxi.us/images/2300830-1.jpg

- Multimeter-Messkabel schwarz auf den Seiten ?? und 73

  http://jto.tinxi.us/images/2300829-2.jpg

- RGB-LED auf Seite 78

  http://www.tandyonline.co.uk/media/catalog/product/cache/1/image/
  9df78eab33525d08d6e5fb8d27136e95/r/g/rgb-led.png

- Analoges Messgerät auf Seite 86

  http://upload.wikimedia.org/wikipedia/commons/5/59/AMM_Skalen_nl.jpg

- Schalter on/off auf Seite 86

  http://www.mx-forum.de/userpix/5736_on_off_schalter_1.jpg

- Servomotor auf Seite 90

  http://cdn.pollin.de/article/xtrabig/X820080.JPG

- Servomotor auf Seite 90

  http://www.rc-forum.de/drupal/userfiles/image/rc_berichte/autoren/RobertvonGoess/53_Servo_gr.jpg

- Stromadapter auf Seite 93

  http://www.exp-tech.de/images/product_images/original_images/10811-01.jpg

- ULN2803a auf Seite 96

  http://www.conrad.de/medias/global/ce/1000_1999/1700/1760/1764/176451_BB_00_FB.EPS_1000.jpg

- Schrittmotor auf Seite 98

  https://www.modmypi.com/

- Minecraft Pi Edition auf Seite 104

  http://d3bgui9r1m2641.cloudfront.net/wp-content/uploads/2012/12/cropped-Pi-Edition-header1.jpg

Kapitel 6

Haftung für Links zu Webseiten

In diesem Mdoul sind Querverweise (Links) zu Webinhalten fremder Betreiber angegeben. Auf die Inhalte dieser Webseiten haben wir keinen Einfluss.

Bei der erstmaligen Angabe haben wir den fremden Inhalt daraufhin überprüft, ob durch ihn eine mögliche zivilrechtliche oder strafrechtliche Verantwortlichkeit ausgelöst wird.
Rechtswidrige Inhalte waren zum Zeitpunkt der Verlinkung nicht erkennbar.

Trotz sorgfältiger inhaltlicher Kontrolle übernehmen wir keine Haftung für die Inhalte externer Links.
Für den Inhalt der verlinkten Seiten sind ausschließlich deren Betreiber verantwortlich.

Passau, im August 2016

Ute Heuer, Wolfgang Pfeffer