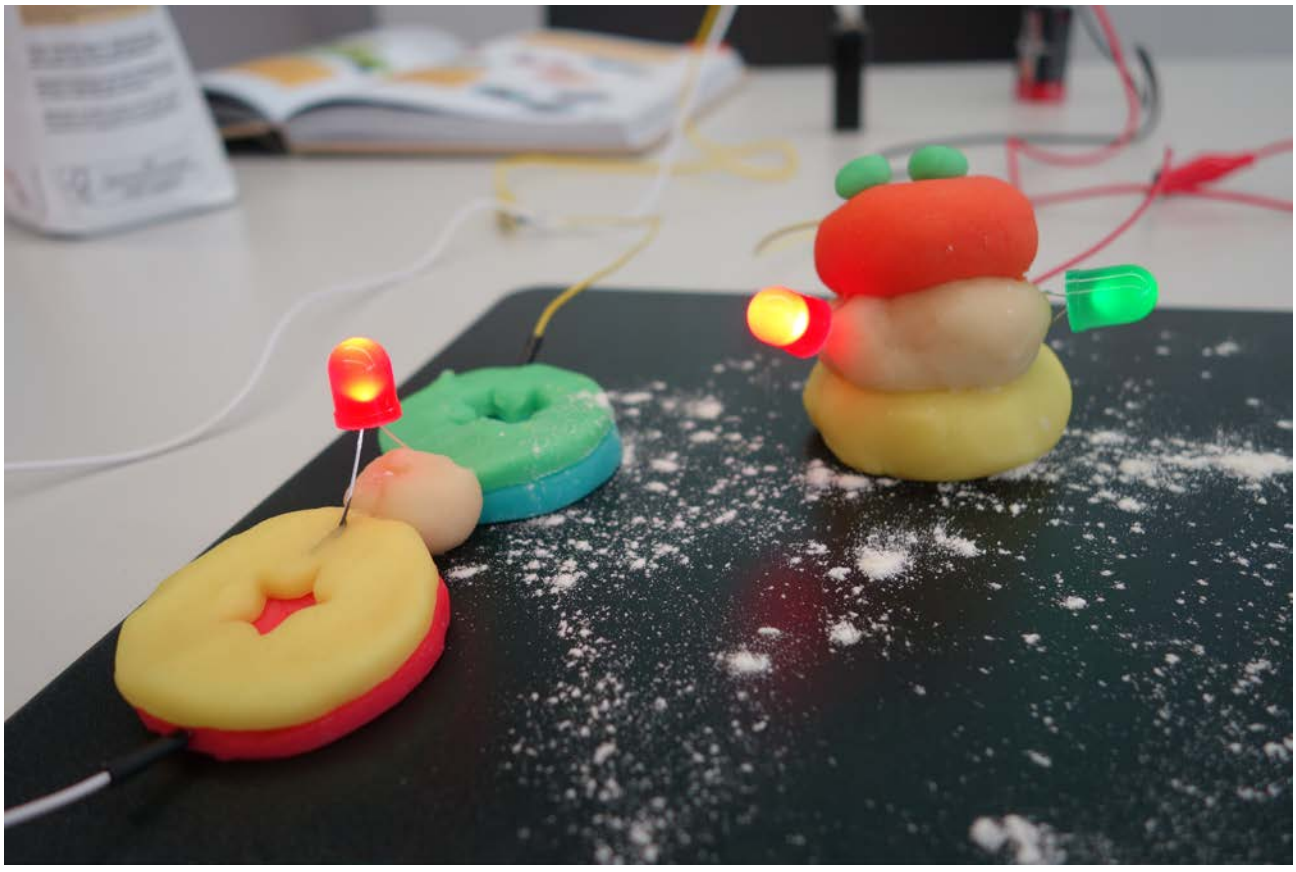
pädagogische hochschule schwyz

Physical Computing – Verbindung der physischen mit der virtuellen Welt



Impressum

Version 2.0 (Juli 2018)

Dr. Dorit Assaf
Dozentin für Informatik
Pädagogische Hochschule Schwyz
www.phsz.ch
dorit.assaf@phsz.ch

Dieses Dokument basiert auf der Version 1.0 (März 2018) von Dorit Assaf, PHZH.

Creative Commons

Namensnennung-Weitergabe unter gleichen Bedingungen

Ursprungsnachweis der Grafiken und Bilder

Bilder:

Titelbild «Leitfähiger Teig mit LEDs». Quelle: Dorit Assaf, Workshop «Mein Guetsliteig schickt mir eine E-Mail!» – alltägliche Materialien mit der digitalen Welt verbinden. UNM-Tagung, PHZH 2016.

Tragbares Transistorradio «Transita» der Firma Nordmende aus den 1960er-Jahren. Quelle: KMJ aus der deutschsprachigen Wikipedia.

Taschenradio vermutllich späte 1960er-/1970er Jahre. Quelle: Ulfbastel aus der deutschsprachigen Wikipedia.

Micro servo. Quelle: oomlout aus der englischsprachigen Wikipedia.

Arduino Uno - R3. Quelle: SparkFun Electronics from Boulder aus der englischsprachigen Wikipedia

The Raspberry Pi 2 (model B). Quelle: Evan-Amos aus der englischsprachigen Wikipedia.

Calliope mini. By Jørn Alraun aus der deutschsprachigen Wikipedia.

Grafiken: Alle Grafiken wurden erstellt von Dorit Assaf

Icons: The Noun Project (https://thenounproject.com)
Arduino Board by Sonja Grapemind, *Battery* by Sergey Demushkin, *LED* by Arthur Shlain, *switch* by Arthur Shlain, *Thermometer* by Hopkins, *Speaker* by Edward Boatman, *Compass* by FakehArtwork, *knob* by Tom Van T Westeinde, *Gears* by icon 54, *Button Click* by andriwidodo, *microprocessor* by Adnen Kadri, *Potentiometer* by Hans, *distance* by Numero Uno, *verbal communication* by Gregor Cresnar, *potentiometer* by Knut M. Synstad, *switch* by Studio Refine, *Resistor* by Studio Refine, *LED* by Victor Bolivar, *Diode LED* by Giovanny, *vibration motor* by Hans, *electric motor* by Arthur Shlain, *Servo motor* by Branis Panos, *Alarm* by Sergey Demushkin, *Resistor* by Bakunetsu Kaito, *brightness* by Hermine Blanquart, *Cards* by Made, *Airplane* by icon 54, *photocell* by Hans, *thumbs up* by Alex Muravev

Inhaltsverzeichnis

1 Zu diesem Heft

1.1 Was ist Physical Computing?

Wenn man an Computer und digitale Geräten denkt, kommen einem wahrscheinlich in erster Linie Laptops, Desktops, Smartphones und Tablets in den Sinn. Dies ist natürlich korrekt, diese Geräte machen aber nur einen kleinen Teil der digital vernetzten Welt aus. So benutzen wir tagtäglich bewusst und unbewusst eine Vielzahl von Computern, sei es im Self-Checkout im Supermarkt, beim Betreten eines Gebäudes, im Auto, im Kino, im Fitnesscenter usw. Wenn wir die Informatik verstehen und die Möglichkeiten von Computern erkunden möchten, müssen wir das gängige Bild eines Computers ablegen und eher von «Computing» sprechen. Mit Computing ist dabei die automatisierte Informationsverarbeitung gemeint, also alle Tätigkeiten, die mithilfe von Algorithmen auf Computersystemen ausgeführt werden. Dabei können diese Computersysteme jede technisch mögliche physische Form annehmen, die für die Tätigkeit notwendig ist: z.B. ein Fitnessarmband, ein Babyphon, elektronische Anzeigen an Haltestellen, Zugangssysteme in Gebäuden, Gamekonsolen, Smart-TVs, Staubsaugerroboter, Analysegeräte in Arztpraxen, Alarmanlagen, moderne Prothesen und Implantate. Computing umfasst den Entwurf und die Entwicklung von Hardware und Software, sowie die Strukturierung und Verarbeitung verschiedener Arten von Informationen. Mit «Computational Thinking» ist die Problemlösekompetenz gemeint, die es im Computing braucht: ein Ziel zu definieren, das Problem in Teilprobleme aufzubrechen und eine Lösung zu implementieren, welche anschliessend getestet und verbessert wird. Die Lösung soll dabei von einem Computer ausführbar sein.

«Physical Computing» ist ein Bereich des Computing, welcher die physische Welt mit der virtuellen Welt von Computern verbindet. Es geht also nicht nur um die Computer selber, sondern auch um die Interaktion mit der physischen Aussenwelt. Als zentrales Konzept gilt der Informationsfluss, der von der physischen Welt in die virtuelle Welt gelangt, dort verarbeitet wird und wiederum in die physische Welt ausgegeben wird. Natürlich funktioniert jeder Computer nach dem «EVA-Prinzip», Eingabe – Verarbeitung – Ausgabe: ein Laptop hat beispielsweise Maus und Tastatur für die Eingabe, einen Prozessor für die Verarbeitung und einen Bildschirm und Lautsprecher für die Ausgabe. Physical Computing geht aber darüber hinaus und befasst sich noch tiefer mit den Eingabe- und Ausgabemöglichkeiten und der Interaktion mit der physischen Welt über Sensoren und Aktoren. Es geht dabei auch hauptsächlich darum, solche interaktiven Computersysteme mit kleinen Computern selber zu entwerfen, zu entwickeln und zu programmieren.

Roboter sind sehr beliebt als didaktisches Mittel im Informatik- und Technikunterricht in der Volksschule. Bezüglich Computerhardware, Sensoren, Aktoren sowie Programmierkonzepten gibt es beim Thema Physical Computing keinen Unterschied zur Robotik – in beiden Gebieten verbindet man die physische mit der virtuellen Welt. Ob ein Projekt ein Robotikprojekt ist, hängt eher von der Anwendung des Endprodukts ab.

Roboterprojekte befassen sich eher mit Systemen, die sich autonom in der Umwelt orientieren und/oder physische Arbeit für den Menschen verrichten. Oftmals sind die Produkte Roboterarme, kleine autonome Fahrzeuge und Drohnen, sowie humanoide Roboter. In der Roboterprogrammierung liegt der Schwerpunkt häufig bei der Lösung von Problemen der autonomen Navigation des Roboters und der künstlichen Intelligenz. Produkte im Bereich des Physical Computings können alles Mögliche sein: eine automatische Bewässerungsanlage für die Zimmerpflanze, ein Fitnessarmband, eine interaktive Kunstinstallation, eine automatische Zugangskontrolle für die Katzentür, E-Textilien, ein interaktiver Geburtstagskuchen usw. Die benötigten Kompetenzen in Programmierung sowie Hardware- und Software-Fachwissen sind im Bereich des Physical Computings und in der Robotik identisch. Wer sich also mit Physical Computing auseinandersetzt, kann problemlos in das Thema Robotik einsteigen und umgekehrt.

1.2 Warum eignet sich Physical Computing im Unterricht?

Das Ziel des Moduls «Medien und Informatik» des Lehrplans 21 ist es, die digital vernetzte Welt zu verstehen, zu gestalten, zu nutzen und ihre Auswirkungen auf die Gesellschaft zu verstehen. Im Teilbereich Informatik geht es nicht darum, Informatikerinnen und Informatiker für eine spätere Berufslaufbahn auszubilden, sondern eine Grundbildung im Sinne der Volksschule zu vermitteln. Die erworbenen Kompetenzen sollen die Schülerinnen und Schüler für die Zukunft im digitalen Alltag und im Berufsleben fit machen. Das umfasst das Verständnis der Informations- und Kommunikationstechnologien, welche sie bewusst und unbewusst in ihrem Alltag nutzen. Und dazu gehören neben «klassischen» Computern auch alle anderen vernetzten digitalen Geräte, welche vermehrt durch Sensoren und Aktoren mit der Umwelt interagieren, sowie Daten sammeln, speichern und verarbeiten.

Physical Computing eignet sich hinsichtlich verschiedener Aspekte als Thema im Unterricht. Wie in der Robotik sind die Aktivitäten «hands-on», konstruktiv, projektorientiert und fördert eine Vielfalt von Kompetenzen. Sie eignen sich, um die oftmals abstrakten Inhalte der Informatik durch das Greifbarmachen besser zu vermitteln. Es geht nicht «nur» um die Programmierung in einem virtuellen Kontext, sondern bezieht die physische Welt sowie die Interaktion mit dem Nutzer mit ein. Und dabei muss man sich auch mit Aspekten der Elektronik, Gestaltung, Physik, Benutzerfreundlichkeit usw. auseinandersetzen. Aber auch hinsichtlich der Informatik deckt das Thema Physical Computing viele Kompetenzen ab. Da man sich mit dem Informationsfluss von der physischen Welt über die digitale zurück zur physischen Welt auseinandersetzen muss, wird das Verständnis von Digitalisierung, Codierung, Datenstrukturen, Algorithmen und Computern allgemein gefördert.

Die Projekte in Physical Computing sind sehr vielfältig und befinden sich an den Schnittstellen zu anderen Disziplinen. So sind Projekte im Kontext des textilen und technischen Gestaltens sehr beliebt, ebenso wie Projekte in Kunst und Naturwissenschaften. Auch im Zusammenhang mit Sprachunterricht gibt es viele Projektbeispiele. So kann Physical Computing dazu beitragen, Informatik fächerintegriert zu unterrichten. Und schliesslich macht das das Arbeiten in Physical Computing auch einfach nur Spass!

1.3 Warum dieses Heft?

In den letzten Jahren wurden zahlreiche kostengünstige didaktische Tools für den Einsatz im Unterricht auf der Volksschule entwickelt, welche einen niederschwelligen Zugang zum Thema Physical Computing ermöglichen, wie z.B. micro:bit, CodeBug, Calliope, OxoCard, Arduino, RaspberryPi, LittleBits, LilyPad, LEGO WeDo/Mindstorms usw. Dabei handelt es sich um kleine, programmierbare Computerboards, an die zahlreiche Sensoren und Aktoren angeschlossen werden können.
Zu jedem dieser Tools existieren mehr oder weniger umfangreiche Tutorials. Diese Tutorials fokussieren aber meistens auf die ersten Schritte: wie man das Tool in Betrieb nimmt, Code programmiert und auf das Board hochlädt. Dazu gibt es zahlreiche kreative Projektideen mittels Schritt-für-Schritt-Anleitungen.
Diese Tutorials ermöglichen zwar einen geleiteten, niederschwelligen Einstieg, um in kurzer Zeit ein funktionierendes Produkt zu entwickeln. Die Problematik ist jedoch, dass die Nutzer oftmals den Programmcode und die Elektronik nicht wirklich verstehen und alles «nur» streng nach Anleitung zusammenstecken bzw. zusammenklicken. Ohne die Grundlage, was ein Computer ist, was Sensoren bzw. Aktoren sind, was für Signale sie zurückliefern bzw. benötigen, was die Programmierkonzepte sind, ist es sehr schwierig, sich über die Tutorials hinaus zu bewegen. Wichtige Aspekte wie der Problemlöseprozess – wie man von der Projektidee zum fertigen Produkt kommt – werden oftmals ungenügend angesprochen. Ohne diese Grundlagen und nur über Schritt-für-Schritt-Tutorials werden Lehrpersonen die benötigten Kompetenzen und das Konzeptwissen nur mit einem erheblichen Zusatzaufwand erlangen, um diese Tools selbst erfolgreich im Unterricht einzusetzen. Aus diesem Grund widmet sich dieses Heft den Konzepten zum Thema Physical Computing, welche all diesen didaktischen Tools zugrunde liegen.

1.4 Für wen ist dieses Heft?

Ganz im Sinne des GMI21 Projektes ist das hauptsächliche Zielpublikum:

- Hochschullehrende im Bereich Medien und Informatik an Pädagogischen Hochschulen der Aus- und Weiterbildung.
- Studierende an Pädagogischen Hochschulen / Lehrpersonen der Volksschulen.

Natürlich richtet sich dieses Heft auch an alle anderen Interessierten, die sich mit dem Thema Physical Computing im Unterricht auseinandersetzen möchten.

Das Heft kann einerseits linear durchgelesen werden, eignet sich aber auch als Nachschlagewerk. Kapitel 2 umfasst das Konzeptwissen zu Physical Computing, mit Querverweisen auf die praktischen Übungen von Kapitel 4. Die Übungen wurden mit den beiden Tools micro:bit und Calliope erstellt, zwei zurzeit sehr beliebten Produkten im Einsatz auf der Volkschule. Die in diesem Heft vermittelten Konzepte gelten aber auch für alle anderen Produkte, sodass ein Transfer zu einem anderen didaktischen Tool gut möglich sein sollte. Kapitel 3 befasst sich schliesslich mit den didaktischen Fragestellungen beim Einsatz solcher Tools im Unterricht auf der Volksschule.

1.5 Zu erwerbende Kompetenzen

Die fachlichen Kompetenzen der Lehrpersonen sind identisch mit denen der Schülerinnen und Schüler im Modul «Medien und Informatik» des Lehrplans 21.

Die Lehrpersonen (Schülerinnen und Schüler)...

- können logische Operatoren verwenden (und, oder, nicht) (MI.2.1.i).
- können selbstentdeckte Lösungswege für einfache Probleme in Form von lauffähigen und korrekten Computerprogrammen mit Schleifen, bedingten Anweisungen und Parametern formulieren (MI.2.2.g).
- können selbstentwickelte Algorithmen in Form von lauffähigen und korrekten Computerprogrammen mit Variablen und Unterprogrammen formulieren (MI.2.2.h).
- kennen die wesentlichen Eingabe-, Verarbeitungs- und Ausgabeelemente von Informatiksystemen und können diese mit den entsprechenden Funktionen von Lebewesen vergleichen (Sensor, Prozessor, Aktor und Speicher) (MI.2.3.l).

Für einen erfolgreichen Unterricht sind zusätzlich zu den oben genannten fachlichen Kompetenzen folgende fachdidaktische Kompetenzen zu erwerben:

Die Lehrpersonen...

- kennen Aktivitäten und Tools zum Thema «Physical Computing» für die Umsetzung der oben aufgeführten Kompetenzen des Lehrplans 21.
- kennen Möglichkeiten, ihren Unterricht so zu organisieren, dass die Inhalte des Themendossiers vermittelt werden.
- kennen Ressourcen und weiterführende Unterlagen zum Thema «Physical Computing».

2 Zum Thema

2.1 Ein niederschwelliger Zugang zur digital vernetzten Welt

Wie im Kapitel 1.1 bereits erwähnt, benutzen wir tagtäglich bewusst und unbewusst eine Vielzahl von Computern. Die meisten dieser Geräte sind nicht so offensichtlich als Computer erkennbar, wie dies bei Laptops oder Smartphones der Fall ist. Diese «unsichtbaren» Computer sind in unseren alltäglichen Geräten eingebaut, wie z.B. in der Kaffeemaschine, in der Waschmaschine, im Auto, im Laserdrucker, im batteriebetriebenen Spielzeug, in der Hausautomation usw. Aus diesem Grund werden sie auch als «eingebettete Systeme» (embedded systems) bezeichnet. Sie bestehen aus kostengünstigen Mini-Computern und finden sich in Systemen wieder, wo die Ressourcen bezüglich Rechenleistung, Strombedarf, Grösse, Hardwarekosten usw. eine Rolle spielen. Mikrocontroller sind solche Mini-Computer.
Früher benötigte man sehr viel Know-How, wenn man sich mit Mikrocontrollern beschäftigen wollte. Die kleinen Chips sind zwar sehr günstig und können einfach beschafft werden. Um einen Mikrocontroller jedoch nur schon zum Laufen zu bringen, benötigt man eine Vielzahl von weiteren elektronischen Komponenten, die man in einer Schaltung für die Stromversorgung, den Takt, die Reset-Funktion usw. aufbauen muss. Damit der Mikrocontroller auch noch etwas Sinnvolles macht, muss er programmiert werden. Dazu muss man weitere Komponenten für die Programmierschnittstelle anschliessen. Auf der Softwareseite muss man sich in die Programmierumgebung des Herstellers des Mikrocontrollers einarbeiten. Die Programmierung ist sehr «low-level» und anspruchsvoll, meistens in der Programmiersprache C/C++ (oder auch Assembler), worin die Register des Mikrocontrollers mit Bitoperationen gesetzt werden. Um dies zu tun, muss man sich mit dem Datenblatt des Mikrocontrollers auseinandersetzen, das sehr umfangreich und für Laien schwer verständlich ist. Die Welt der Mikrocontroller und somit der interaktiven Systeme mit Sensoren und Aktoren war bis vor kurzem eher Fachpersonen und Crack-Hobbyisten vorbehalten.

Das Arduino-Projekt aus Italien brachte im Jahre 2005 einen Stein ins Rollen (Abbildung 1). Die Idee war es, Mikrocontroller für Künstler, Designer, Hobbyisten und Studierende niederschwellig zugänglich zu machen. Das Arduino-Board beherbergt einen Mikrocontroller und enthält alle Komponenten, um diesen zum Laufen zu bringen und verfügt über eine USB-Schnittstelle für die Programmierung. Dazu kommt eine Programmierumgebung sowie eine Library mit vereinfachten Funktionen. Der Nutzer muss sich dadurch über Register und Bitoperationen keine Gedanken mehr machen. Die Schaltpläne des Arduino-Boards und die Software sind Open-Source. Dies ermöglichte die Weiterentwicklung von Seiten der Open-Source-Community. In wenigen Jahren sind die Entwicklungen zu Arduino-kompatiblen Erweiterungsplatinen (shields) und Software Libraries explodiert. Ein weiterer Meilenstein war die Entwicklung des Raspberry Pi im Jahre 2012. Dies ist ein kostengünstiger und vollwertiger Personal Computer im Kreditkartenformat mit Linux-Betriebssystem inkl. Pins für den Anschluss von Sensoren und Motoren. Dadurch lassen sich webbasierte Entwicklungen leichter bewerkstelligen als auf einem Arduino. Der Erfolg dieser Projekte hat die neuen Entwicklungen von Physical Computing Plattformen speziell für die Schule inspiriert, wie z.B. der BBC micro:bit (2015) und Calliope mini (2016) (Abbildung 1).

Abbildung 1 – Physical Computing Plattformen (von links nach rechts): Arduino Uno, Raspberry Pi, BBC micro:bit, Calliope mini.

2.2 Digitalisierung

«Die Digitalisierung ist im vollen Gange», «Der digitale Wandel», «Die Digitalisierung der Schule». Überall begegnet man dem Ausdruck «Digitalisierung». Ebenso werden die Begriffe «analog» und «digital» in verschiedenen Zusammenhängen verwendet. Im Alltag sind oftmals Aussagen zu hören, wie z.B.: «Habt ihr euch in der analogen oder digitalen Welt kennengelernt?», «Ich lese das Buch immer noch gerne auf analoge Weise», «Die Tonqualität von Musik auf einem analogen Plattenspieler ist einfach unschlagbar!» usw. Doch was genau bedeutet analog und digital? Sind das Gegensätze? Was passiert bei der Digitalisierung?

Mit Digitalisierung ist heute im Allgemeinen die Veränderungen von Prozessen in unserem Leben durch die zunehmende Nutzung vernetzter digitaler Geräte gemeint. Dieser Wandel wird auch als «digitale Transformation» bezeichnet und durchdringt alle Lebensbereiche, wie Bildung, Wirtschaft, Kultur, Politik und Gesellschaft. Es gibt keinen Bereich, der sich nicht mit der Digitalisierung auseinandersetzen muss. Auch die Bildungsinstitutionen sind stark von diesem Wandel betroffen. Der Lehrplan 21 mit dem neuen Modul «Medien und Informatik», die Ausrüstung der Schulen mit IT-Infrastruktur sowie die Aus- und Weiterbildung der Lehrpersonen in den Disziplinen Medienbildung und Informatik sind Konsequenzen dieses digitalen Wandels.

2.2.1 Digitalisierung in der Informatik

In der Informatik wird im ursprünglichen und engeren Sinne der Begriff Digitalisierung etwas anders verwendet. Hier wird mit Digitalisierung der Prozess bezeichnet, bei welchem Informationen wie z.B. ein handgeschriebener Text, Schallwellen, ein Bild, eine Temperatur usw. in eine digitale Repräsentation überführt werden. Der Sinn und Zweck der Digitalisierung ist die Bereitstellung digitaler Daten für die anschliessende automatisierte Speicherung, Verarbeitung und Ausgabe durch Computer.
Bei der Digitalisierung können Informationen in beliebiger Form vorliegen. Bei einem klassischen Thermometer beispielsweise wird die Temperatur über die Ausdehnung von Flüssigkeiten, Gasen oder Festkörpern mit bekanntem Ausdehnungskoeffizient gemessen (Abbildung 2). Dabei dient die auf dieses Thermometer geeichte Skala als optisches Hilfsmittel, um die Temperatur abzulesen. Die Ausdehnung der Flüssigkeit ist kontinuierlich, d.h., dass sich der Endpunkt auch zwischen den eingezeichneten Skalenpunkten befinden kann. Beim Ablesen wird eine Temperatur zwischen den unteren und oberen Skalenpunkten angenommen. Wegen dieser stufenlosen Anzeige einer physikalischen Grösse wie der Temperatur wird das klassische Thermometer auch als analoges Messgerät bezeichnet. Analog bedeutet hier kontinuierlich.

Bei der Digitalisierung einer analogen Grösse ist der erste Schritt die «Diskretisierung». Das bedeutet, dass der analoge, stufenlose Wertebereich in diskrete Abschnitte eingeteilt wird. Bei der Skala des Thermometers in Abbildung 2 erfolgt dies im Abstand von 1°C. Wenn der Endpunkt der Ausdehnung der Flüssigkeit nun irgendwo zwischen zwei Skalenpunkten liegt, wird bei der Diskretisierung der Wert des entweder nächsten unteren oder oberen Skalenpunkts zugeordnet bzw. es wird «auf- oder abgerundet». Je gröber die Skala, d.h. je grösser die Abstände bei der Skaleneinteilung, desto verlustbehafteter ist die diskrete Zuordnung der Werte. Die Information, dass die eigentliche Temperatur irgendwo zwischen zwei diskreten Werten liegt, geht dabei verloren. Bei einer feineren Einteilung der Skala mit mehr Zwischenpunkten ist der Verlust geringer. Man spricht dann auch von einer höheren Auflösung der Skala (Abbildung 3).

Der nächste Schritt ist die Digitalisierung, d.h. die Zuordnung der Skalenpunkte zu einem Zahlenwert. In Europa würde dies beim Thermometer in Abbildung 2 einem Wertebereich von -5 bis 50 (°C) entsprechen, in den USA wären die Zahlenwerte der Skalenpunkte 23 bis 122 (°F). Ein digitales Thermometer zeigt somit nur noch den Zahlenwert des nächsten auf- oder abgerundeten Skalenpunktes an (Abbildung 4). Die Auflösung bzw. Feinheit der Skala des digitalen Thermometers bestimmt somit die Qualität der Wiedergabe der analogen Grösse Temperatur.

Abbildung 2 – Ein analoges Thermometer

Abbildung 3 – Skalen mit verschiedener Auflösung.

Abbildung 4 – Ein digitales Thermometer.

Je nach Art der vorliegenden Information wird diese mit einer anderen Methode digitalisiert. Bei analogen Fotoapparaten wird das Bild über die Linse auf lichtempfindlichem Film abgebildet, es entsteht ein Negativ. Die Chemikalien auf dem Farbfilm reagieren dabei auf die eintreffenden Lichtwellen. Der Prozess ist ebenfalls stufenlos, also analog. Bei einer Digitalkamera treffen die Lichtwellen ebenfalls über die Linse auf einen Sensor, der die Bildinformation jedoch digitalisiert. Die einzelnen als Raster angelegten Bildpunkte (Pixel) entsprechen hier der Diskretisierung. Bei der Digitalisierung werden jedem Bildpunkt Zahlenwerte im Rot-Grün-Blau-Farbraum (RGB) zugeordnet. Auch hier gilt: Je höher die Auflösung, also die Anzahl Bildpunkte, desto besser wird das stufenlose, analoge Bild wiedergegeben (Abbildung 5). Bei Digitalkameras liegt die Auflösung im Megapixel-Bereich, also bei Millionen von Bildpunkten.

Abbildung 5 – Ein Bild mit unterschiedlicher Auflösung. Von links nach rechts: 2’457’600 Pixel, 2’730 Pixel, 983 Pixel.

2.2.2 Digitalisierung eines analogen Signals

Im Physical Computing wird die physische mit der digitalen Welt verbunden. Um die physische Welt zu «fühlen», werden Sensoren verwendet. Ein Sensor wandelt physikalische Grössen in elektrische Signale um, meistens in Form einer elektrischen Spannung in Volt (V). Ein Signal ist eine Anzahl dieser Sensormesswerte oder Abtastwerte im Zeitverlauf (Abbildung 6).
Um den analogen, elektrischen Abtastwert für die Weiterverarbeitung im Computer zu digitalisieren, verfügt dieser über einen «Analog-Digital-Wandler» (ADC). Die Qualität der Digitalisierung hängt von zwei Faktoren ab: Die «Bittiefe» und die «Abtastrate».

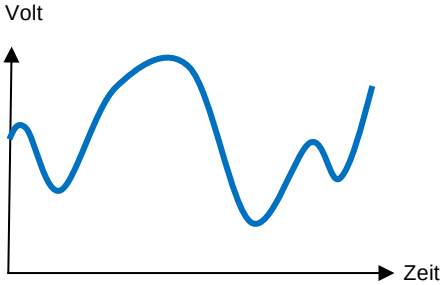


Abbildung 6 – Ein kontinuierliches, analoges Signal.

Bittiefe:
Sie bezeichnet die Auflösung der Skala, durch welche der Abtastwert digitalisiert wird (Abbildung 7). Diese wird in Form einer Zahl angegeben, die den Wertebereich der Skala bestimmt. Wenn die Zahl beispielsweise 100 beträgt, wird die Skala in 100 Skalenpunkten eingeteilt. Wenn das analoge Signal nun einen Wertebereich von 0 bis 5 Volt hat, ist die Genauigkeit des Analog-Digital-Wandlers 0.05 Volt (5V/100). Ein Abtastwert von beispielsweise 2.5 Volt entspricht dann dem digitalen Wert 50 (2.5/0.05), 1.8 Volt einem Wert von 36 (1.8/0.05) usw. Da ein Computer mit dem binären Zahlensystem rechnet, wird die Auflösung des Analog-Digital-Wandlers in Bit angegeben. Bei Mikrocontrollern ist dies häufig 10 Bit, d.h. ein Wertebereich von 0 bis 1023 (2^{10}-1). Die Auflösung ist somit 0.004887 Volt (5V/1023) [1], d.h. dass sich mit der 10-Bit-Auflösung feinere Messwerte unter 4.887 Millivolt nicht mehr unterscheiden lassen.

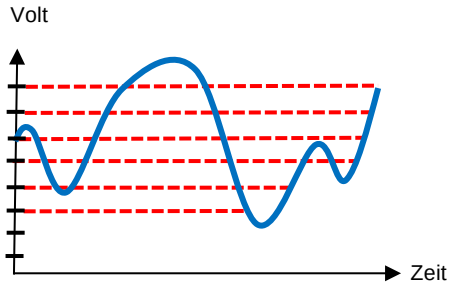


Abbildung 7 – Die Bittiefe entspricht der Auflösung der Skala bei der Digitalisierung des Abtastwerts.

Abtastrate:
Sie bezeichnet die Häufigkeit, mit der das analoge Signal abgetastet wird, also die Auflösung bezüglich des Zeitverlaufs (Abbildung 8). Auch bei einer sehr hohen Bittiefe ist die Qualität der Digitalisierung nicht hoch, wenn die Abtastrate niedrig ist. Die Abtastrate wird als Frequenz in Hertz (Hz) angegeben. Bei einer Abtastrate von beispielsweise 14.7kHz wird der Messwert alle 68 Mikrosekunden digitalisiert (1/14'700). Das Ergebnis der Digitalisierung eines analogen Signals ist eine Reihe von diskreten Zahlenwerten, die im Computer weiterverarbeitet werden können (Abbildung 9).

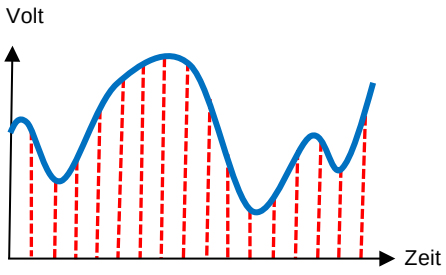


Abbildung 8 – Die Abtastrate ist die Häufigkeit mit der das analoge Signal vom Analog-Digital-Wandler abgetastet wird.

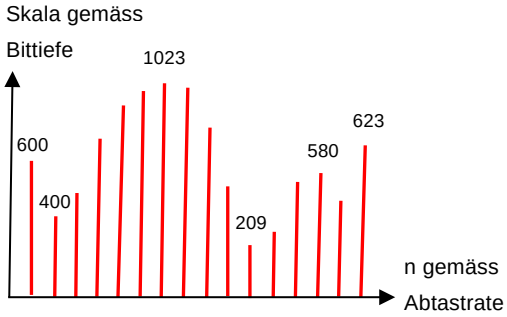


Abbildung 9 – Das Ergebnis der Digitalisierung eines analogen Signals durch einen Analog-Digital-Wandler ist eine Reihe von diskreten Zahlen.

2.2.3 Analoge und digitale Schaltkreise

Digitale Geräte verarbeiten Informationen. Vor dem Computerzeitalter gab es natürlich bereits elektronische Geräte, die ebenfalls Informationen verarbeiteten. Ein klassisches, analoges Radio beispielsweise hat ebenso eine Benutzerschnittstelle für die Wahl der Radiosender, die Einstellung der Lautstärke usw. (Abbildung 11). Es empfängt Informationen in der Form von Radiofunkwellen über eine Antenne und wandelt diese über den Lautsprecher in Schallwellen um. Klassische elektronische Geräte verfügen über einen analogen Schaltkreis, bestehend aus analogen elektronischen Bauteilen wie z.B. Kondensatoren, Widerständen, Spulen, Transistoren, Verstärker, Filter usw. (Abbildung 10). Diese Schaltkreise funktionieren mit analogen Signalen.

[1] Der Einfachheit halber wird hier angenommen, dass die Referenzspannung (AREF) des Analog-Digital-Wandlers gleich der Spannung des Signals ist.

Abbildung 11 – Tragbares Transistorradio "Transita" der Firma Nordmende aus den 1960er Jahren.

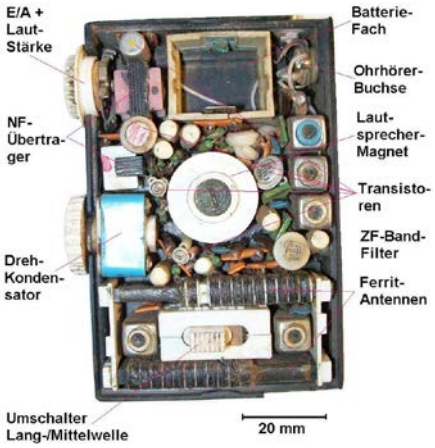


Abbildung 10 – Geöffnetes Taschenradio der 1960er/1970er Jahre. Dieses Radio funktioniert ausschliesslich mit analogen elektronischen Bauteilen.

Digitale Schaltkreise hingegen beinhalten digitale elektronische Bauteile wie Logikgatter, Mikrocontroller, Datenspeicher usw., die mit digitalen Signalen funktionieren. Sobald die Informationsverarbeitung mit Computern bzw. Mikroprozessoren erfolgt, kann man von digitalen Geräten sprechen. In der Realität bestehen digitale Geräte jedoch aus «Mixed-Signal-Schaltungen», d.h. in den Schaltkreisen werden analoge und digitale Bauelemente kombiniert. Die Signale werden durch Analog-Digital-Wandler und Digital-Analog-Wandler hin und her gewandelt. Oftmals bestehen die Benutzerschnittstellen, wie die Drehköpfe am Radio, Mikrofon, Lautsprecher usw. aus analogen Bauteilen bzw. Signalen, da sie mit der physischen Welt in Kontakt treten (siehe Kapitel 2.3). Der Einfachheit halber sprechen wir in diesem Heft aber von digitalen Schaltkreisen, sobald ein Mikrocontroller im Einsatz ist.

Der wichtigste Unterschied bei der Arbeit mit Mikrocontrollern im Gegensatz zu analogen Schaltkreisen ist die Steuerung der Bauteile. Wenn man bereits Erfahrungen mit analogen, elektronischen Schaltkreisen hat, kann dies zunächst etwas verwirrend sein. Abbildung 12(a) zeigt einen klassischen analogen Schaltkreis mit Stromquelle, LED und Schalter. Sobald der Schaltkreis über den Schalter geschlossen wird, leuchtet die LED, da der Strom nun ungehindert von einem Pol der Batterie zum anderen fliessen kann[2]. Das digitale Pendant zu dieser Schaltung zeigt Abbildung 12(b). Die Batterie ist nun am Mikrocontroller angeschlossen. Die LED und der Schalter sind ebenfalls mit separaten Schaltkreisen mit den Pins des Mikrocontrollers verbunden. Wird nun der Schalter betätigt, wird die LED nicht mehr automatisch leuchten. Ausschliesslich der Mikrocontroller kann die LED zum Leuchten bringen, nämlich dadurch, dass er auf den jeweiligen Pin, an dem die LED angeschlossen ist, eine Spannung erzeugt. Nur dann fliesst der Strom durch den Schaltkreis der LED.

Damit der Schalter die LED an- und ausschaltet, muss diese Funktionalität im Mikrocontroller programmiert werden: Der Mikrocontroller «horcht» dann auf dem Pin, an den der Schalter angehängt ist, auf das Schliessen des Schaltkreises. Anschliessend lässt der Mikrocontroller den Strom durch die LED fliessen. Umgekehrt schaltet der Mikrocontroller die LED wieder ab, sobald er detektiert, dass der Kreislauf am Pin des Schalters wieder geöffnet ist.

Bei einem ganz einfachen Schaltkreis wie in diesem Beispiel – Schalter steuert LED – wirkt die digitale Schaltung mit dem Mikrocontroller übertrieben. Möchte man jedoch das Verhalten der Schaltung etwas

[2] Der Einfachheit halber wurde hier der zusätzlich benötigte Widerstand in der Schaltung weggelassen.

variieren, wird es mit rein analogen Schaltkreisen schnell komplex. So benötigt man bereits weitere Bauteile, wenn man nur die Funktion umkehren möchte: ein geöffneter Schalter lässt die LED leuchten, oder die LED soll blinken. Mit einem Mikrocontroller kann man dies mit ein paar Zeilen Programmcode erzielen, die Schaltung muss dabei gar nicht verändert werden.

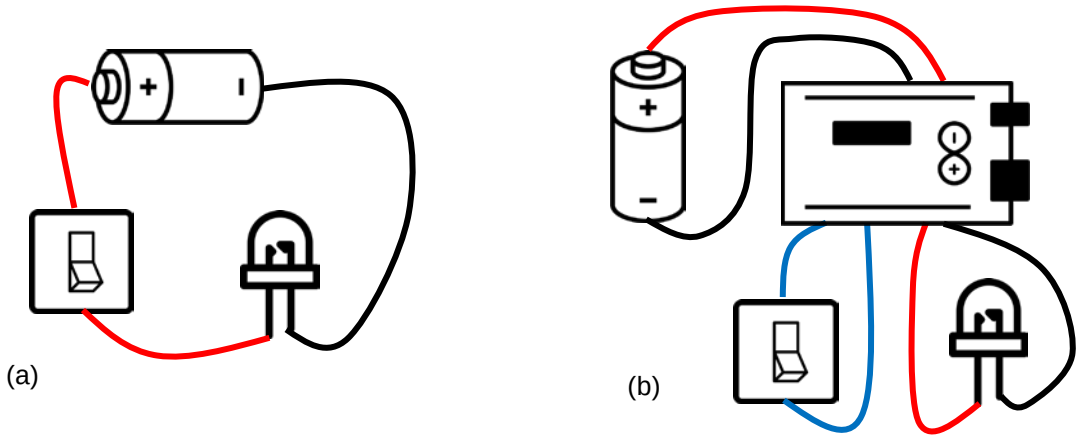


Abbildung 12 – Unterschied zu analogen und digitalen Schaltkreisen: (a) Ein klassischer analoger Schaltkreis mit Stromquelle, LED und Schalter. (b) Ein digitaler Schaltkreis mit Stromquelle, Mikrocontroller, Schalter und LED.

2.2.4 Analoge vs. digitale Welt – physische vs. virtuelle Welt

Der Ausdruck analog wird umgangssprachlich oft mit der physischen Welt in Zusammenhang gebracht. Dies liegt daran, dass viele Prozesse in unserer Welt analog ablaufen. Unsere Sinne, unser Gehirn, unser Organismus funktionieren mit analogen, stufenlosen biochemischen Prozessen. Ebenso definieren analoge, physikalische Gesetze unsere Umwelt, das Wetter, das Universum usw. Klassische, nicht-digitale Medien werden auch als analoge Medien bezeichnet. Wie im vorhergehenden Kapitel beschrieben, beinhalten auch digitale Geräte analoge Komponenten. Auch Computer und Computernetze sind durchaus physisch in unserer Welt vorhanden. Aus diesem Grund ist es nicht ganz korrekt, die analoge Welt als physische Welt zu bezeichnen.
Beim Physical Computing wird die physische mit der virtuellen Welt verbunden. Mit der virtuellen Welt ist diejenige Welt gemeint, die ausschliesslich in Computersystemen in Form von Daten existiert. Durch den Einsatz von Sensoren und Aktoren wird diese virtuelle Welt mit der physischen verbunden.

2.3 Mit der physischen Welt in Kontakt treten

Wie bereits in Kapitel 2.1 beschrieben ist ein Mikrocontroller ein allgegenwärtiger Mini-Computer, der in verschiedensten Geräten eingebettet ist. Er ist ein «integrierter Schaltkreis» (IC), der eine quadratische oder rechteckige Bauform aufweisen kann und aussieht wie ein schwarzer Kunststoffchip mit «Beinchen» (Abbildung 13). Die Beinchen sind die Pins des Mikrocontrollers, an denen die weitere Schaltung der Platine aufgebaut wird. Jeder Pin besitzt verschiedene Funktionen. Über die einen wird die Grundversorgung des Mikrocontrollers gewährleistet, mit Funktionen wie z.B. Stromversorgung, Taktgeschwindigkeit, Reset usw. Weitere wichtige Pins sind die «GPIO-Pins» (General Purpose Input Output), über die der Nutzer frei

verfügen kann. So lassen sich beispielsweise Sensoren und Aktoren anschliessen oder Kommunikationsschnittstellen nutzen. Die jeweilige Funktion der GPIO-Pins wird über die Programmierung vom Nutzer bestimmt. Die häufigsten Funktionen und deren Pinbezeichnungen sind in Tabelle 1 und Tabelle 2 abgebildet.

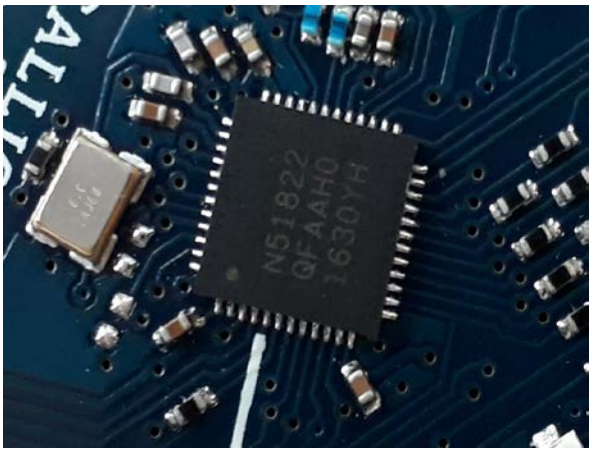


Abbildung 13 – Die Pins eines Mikrocontrollers haben verschiedenste Funktionen, viele davon können vom Nutzer programmiert werden.

Der Nutzer bestimmt, wie die GPIO-Pins in der Schaltung verwendet werden sollen. Ob für ein GPIO-Pin die Funktion Input oder Output zu wählen ist, muss immer aus der Sicht des Mikrocontrollers betrachtet werden: Wenn der Mikrocontroller Informationen von aussen enthalten soll (z.B. über Sensoren), ist dies ein **Input**. Wenn er hingegen etwas ausserhalb ansteuern soll, ist es ein **Output** (z.B. bei Aktoren). Pins wie VCC, GND, AREF, CLK usw. sind fix vergeben, da diese Pins für den Betrieb des Mikrocontrollers benötigt werden. Über alle anderen GPIO-Pins kann frei verfügt werden (Abbildung 14).

Jede Komponente benötigt mindestens zwei Pins, da es immer ein geschlossener Schaltkreis ist. Es ist dabei zu beachten, dass nur bestimmte GPIO-Pins gewisse Funktionen zur Verfügung stellen. Wenn eine bestimmte Kommunikationstechnologie verwendet werden soll (z.B. I^2C, UART, SPI), so sind es nur ganz bestimmte Pins, die über diese Funktion verfügen. Wenn ein Analog-Digital-Wandler oder die analoge Output-Funktion PWM (Pulsweitenmodulation) benutzt wird, muss beim Aufbau der Schaltung auch geprüft werden, welche der Pins diese Funktionen zur Verfügung stellen und welche bereits mit einer anderen Funktion besetzt sind. Gleichzeitig muss bei der Programmierung auch explizit definiert werden, wofür welcher Pin verwendet wird. Der Mikrocontroller detektiert nicht von selber, ob an seinem Pin ein Sensor oder Aktor angehängt ist. Wenn vom Nutzer nichts über die Programmierung definiert wurde, bleiben die GPIO-Pins inaktiv.

Tabelle 2 - Programmierbare GPIO Pins

Funktion	**Typische Pinbezeichnung**
Digitaler Input	GPIO oder D
Digitaler Output	GPIO oder D
Analoger Input über Analog-Digital-Wandler	ADC oder A
Analoger Output	PWM
Serielle Kommunikation UART (TTL serial)	RX/TX
I^2C Kommunikationsbus	SCL/SDA
SPI Kommunikation	MISO/MOSI/SS/ SCK
usw.	

Tabelle 1 – Fixe Pins für die Grundversorgung des Mikrocontrollers

Funktion	**Typische Pinbezeichnung**
Stromversorgung	VCC
Masse	GND
Referenzspannung Analog-Digital-Wandler	AREF
Reset-Pin, startet den Mikrocontroller neu	RESET
Externer Taktinput über einen Quarz	CLK1/CLK2
usw.	

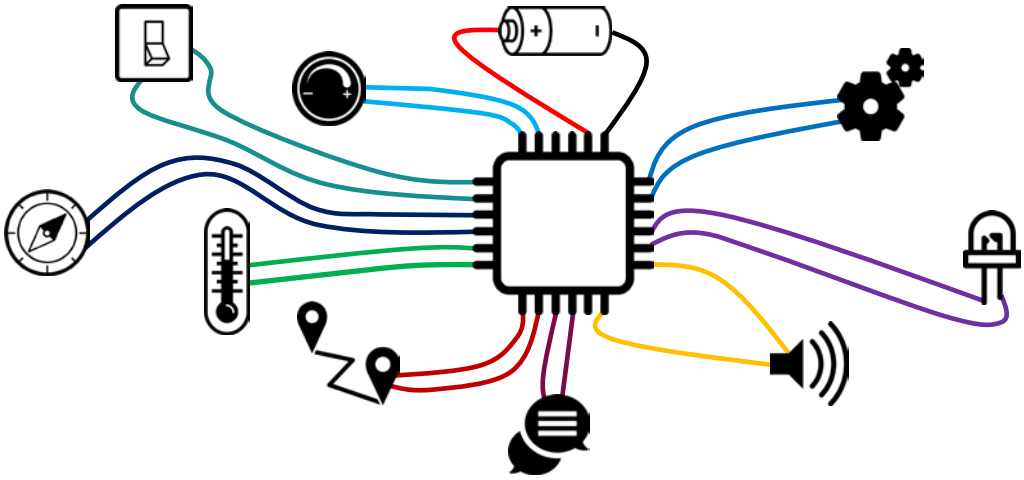


Abbildung 14 – An die Pins eines Mikrocontrollers können die verschiedensten Komponenten angehängt werden, um mit der physischen Welt zu interagieren. Dabei benötigt jede Komponente mindestens zwei Pins, um einen geschlossenen Stromkreislauf aufzubauen. Über die Programmierung definiert man, welche Pins für welche Funktion zuständig sind.

2.3.1 Sensoren

Sensoren übermitteln dem Mikrocontroller Informationen über die Aussenwelt. Aus diesem Grund sind Sensoren **Inputs**. Sensoren wandeln physikalische Grössen wie z.B. Temperatur, Druck, Magnetfeld, Schallwellen usw. in elektrische Signale um. Dabei werden zwei Arten von Signalen unterschieden: analoger Input und digitaler Input.

Analoger Input:
Wie bereits im Kapitel 2.2.2 erwähnt, liefert ein analoges Signal kontinuierliche, stufenlose Messwerte (Abbildung 15). Dieses Signal, z.B. in Form einer Spannung von 0V–5V, wird über den Analog-Digital-Wandler des Mikrocontrollers digitalisiert. Je nach Auflösung (Bittiefe) wird die Eingangsspannung in eine diskrete Zahl abgebildet. Typische analoge Inputs sind: Potentiometer (verstellbarer Widerstand), Temperatursensor, Gyroskop (Drehgeschwindigkeit), Accelerometer (lineare Beschleunigung), Kompass, Infrarotsensor, Mikrofon usw.

Für analoge Sensoren gilt:

- ✓ Analoger Input
- ✓ Müssen an einem GPIO-Pin mit Analog-Digital-Wandler angeschlossen werden.
- ✓ Haben einen kontinuierlichen Wertebereich von z.B. 0 bis 1023 (je nach Auflösung).

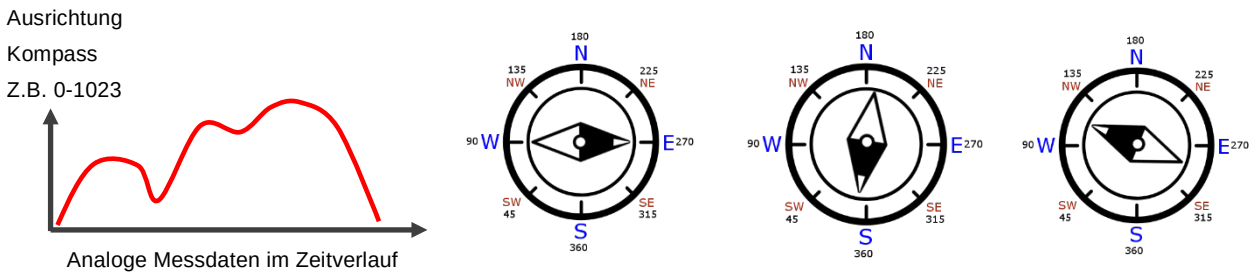


Abbildung 15 – Analoger Input am Beispiel eines Kompasses. Die Messwerte haben einen Wertebereich von 0-360° (wenn bereits auf den Winkel geeicht wurde) oder sonst z.B. die 10 Bit Auflösung von 0-1023.

Zum Ausprobieren:

micro:bit/Calliope Challenge Nr. 5 – Einen verstellbaren Widerstand benutzen.
micro:bit/Calliope Challenge Nr. 6 – Ein Licht dimmen.
micro:bit/Calliope Challenge Nr. 7 – Einen Motor steuern.
micro:bit/Calliope Challenge Nr. 10 – Den Kompass benutzen.
micro:bit/Calliope Challenge Nr. 11 – Die Helligkeit messen.
micro:bit/Calliope Challenge Nr. 12 – Den Lagesensor benutzen.
micro:bit/Calliope Challenge Nr. 13 – Die Temperatur messen.

Digitaler Input:

Digitale Inputs teilen sich wiederum in zwei Untergruppen. Die eine Gruppe sind analoge Sensoren, die aber keine kontinuierlichen Messwerte liefern, sondern als Information nur zwei Zustände haben (0V oder 5V). Solche Sensoren werden als digitale Inputs bezeichnet. Sie benötigen keinen Analog-Digital-Wandler, da ein digitales Gerät wie der Mikrocontroller bereits selber mit den beiden Spannungszuständen von 0V oder 5V[3] arbeitet. Ein digitaler Input kann also direkt an den Mikrocontroller angeschlossen werden. Die beiden Zustände werden im Mikrocontroller als die Zahlen 0 oder 1 repräsentiert. Typische digitale Inputs sind Tasten: eine Taste kann entweder im Zustand gedrückt oder nicht gedrückt sein, einen Wert dazwischen gibt es nicht (Abbildung 16).

Für analoge Sensoren, die ein digitales Signal liefern, gilt:

- ✓ Digitaler Input
- ✓ Können an jedem GPIO Pin direkt angeschlossen werden.
- ✓ Haben einen Wertebereich von zwei Zahlen, 0 oder 1.

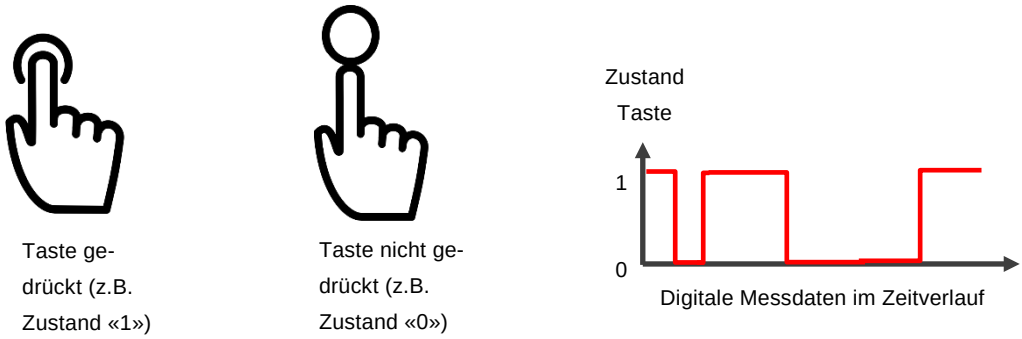


Abbildung 16 – Digitaler Input am Beispiel einer Taste. Der Zustand ist entweder gedrückt oder nicht gedrückt bzw. 5V oder 0V und wird entweder als 1 oder 0 im Mikrocontroller repräsentiert.

Zum Ausprobieren:

micro:bit/Calliope Challenge Nr. 2 – Die Tasten A und B benutzen.
micro:bit/Calliope Challenge Nr. 3 – Die Tasten A und B steuern das Licht.
micro:bit/Calliope Challenge Nr. 4 – Eine Taste steuert das Licht.
micro:bit/Calliope Challenge Nr. 9 – Emojis/Farben mit der Fingerspitze verändern.

[3] Obwohl viele Mikrocontroller-Boards für die Schule über 3.3V laufen, wird hier der Einfachheit halber 5V als Beispielspannung gewählt.

Die zweite Gruppe digitaler Inputs sind digitale Sensoren. Das sind Sensoren, die bereits etwas komplexer sind als die üblichen analogen Bauteile. Sie sind bereits auf einer kleinen Platine verbaut und liefern deswegen die Messwerte in Form von komplexeren digitalen Signalen[4]. Diese Signalarten können variieren. So kann ein Ultraschall-Distanzsensor seine Messwerte in Form von digitalen Messdaten im Zeitverlauf liefern, ähnlich wie bei der Taste in Abbildung 16, mit dem Unterschied, dass die gemessene Distanz des Sensors über die Zeitspanne übermittelt wird, in der das Signal auf «high» geschaltet ist (Abbildung 17). Ein Lagesensor, der üblicherweise bereits mit einem Gyroskop, Beschleunigungssensor und Kompass bestückt ist, liefert alle Messwerte der gesamten Sensorengruppe beispielsweise über die digitale Kommunikationsschnittstelle I²C. Es gibt sehr viele dieser digitalen Sensoren (Abbildung 18). Um herauszufinden, wie die Messwerte ausgelesen werden sollen, muss das Datenblatt des Sensors konsultiert werden. Oftmals sind aber auch die Elektroden dieser kleinen Platinen so beschriftet, dass man auf Anhieb sieht, auf welche Art und Weise der Sensor seine Messwerte übermittelt. Am häufigsten wird der Kommunikationsbus I²C verwendet. Sein Vorteil liegt darin, dass er nur die zwei Pins SDA/SCL am Mikrocontroller besetzt und über hundert I²C-Sensoren an nur diesen beiden Pins angehängt werden können.

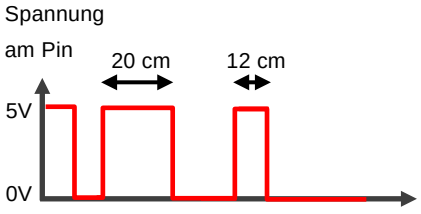


Abbildung 17 – Ein digitaler Input eines digitalen Distanzsensors. Die Zeitspanne, in der das Signal auf «high» (5V) geschaltet ist, bestimmt die gemessene Distanz. Wie genau die Zeitspanne mit der Distanz in cm im Verhältnis steht, muss dem Datenblatt des Sensors entnommen werden.

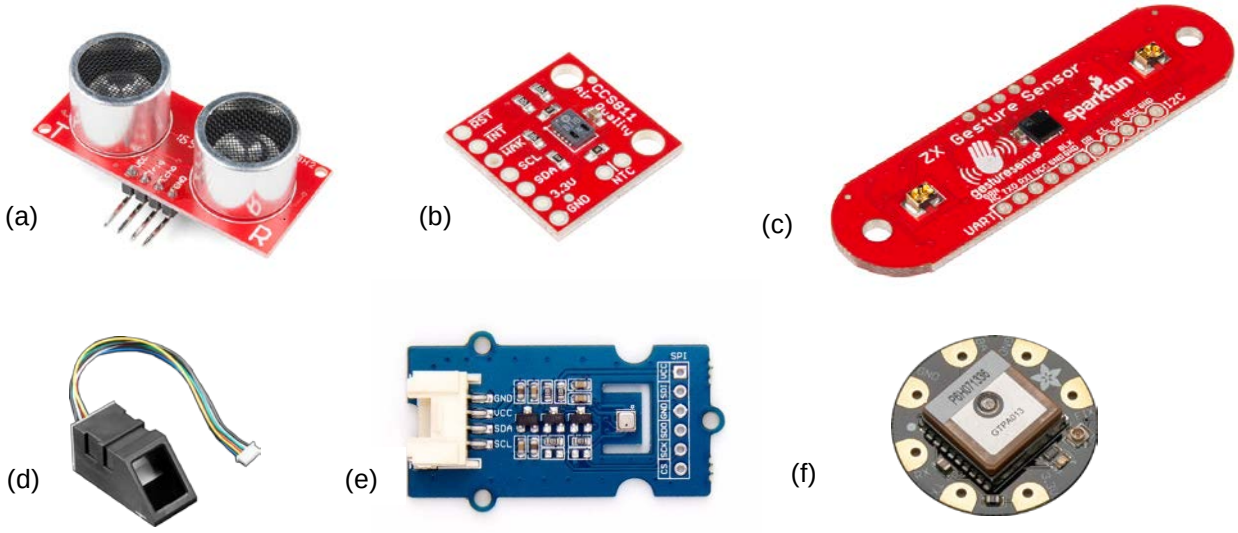


Abbildung 18 – Verschiedene Sensoren mit unterschiedlichen Signalen. Oftmals kann die Art des digitalen Inputs der Pinbezeichnung an den Elektroden entnommen werden. (a) Der Ultraschall Distanzsensor liefert gemäss Datenblatt am Pin Echo den Messwert gemäss Abbildung 17. (b) Ein Luftqualitätssensor, der flüchtige organische Verbindungen misst. Die Pins SCL/SDA verraten, dass diese Messwerte über I²C übermittelt werden. (c) Ein Sensor, der einfache Gesten erkennt. Die Pins UART und I²C verraten, dass die Werte entweder über die serielle Kommunikation oder I²C übermittelt werden können. (d) Ein Fingerabdrucksensor. Dem Datenblatt ist zu entnehmen, dass es sich um eine serielle Kommunikationsschnittstelle handelt. (e) Ein Barometer, Temperatur- und Feuchtigkeitssensor zugleich. SDA/SCL sind klar I²C Schnittstellen. (f) Ein GPS-Modul für E-Textilien. Die Pins RX/TX verraten, dass es sich um eine serielle Kommunikationsschnittstelle handelt.

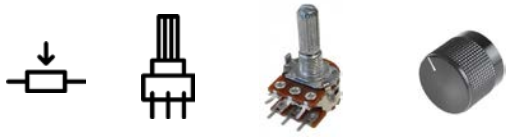
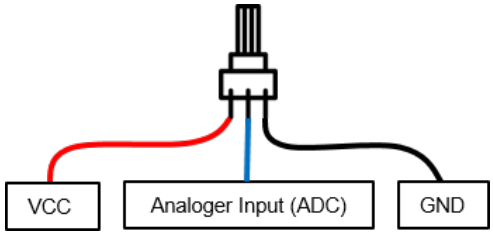
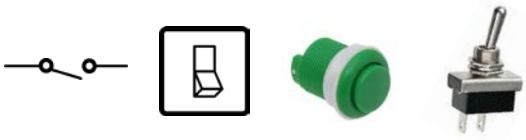
[4] Der Einfachheit halber wird hier nicht erwähnt, dass diese digitalen Sensoren im Grunde genommen auch aus analogen Bauteilen bestehen. Erst die Schaltung der Platine macht diese Sensoren zu digitalen Sensoren.

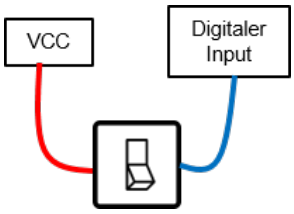
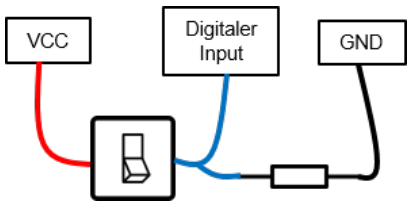
Für komplexere digitalen Sensoren gilt:

- ✓ Digitaler Input
- ✓ Kommunikationsschnittstelle kann variieren, deshalb Datenblatt konsultieren (UART, I²C usw.)
- ✓ Können nur an den GPIO-Pins angeschlossen werden, welche die jeweilige Kommunikationsschnittstelle zur Verfügung stellen.
- ✓ Benötigen eine bestimmte Betriebsspannung, die unbedingt beachtet werden muss. Ein 3.3V-Sensor darf nicht an einen 5V-Pin angehängt werden! Umgekehrt könnte ein 5V-Sensor bei 3.3V nicht korrekt funktionieren.

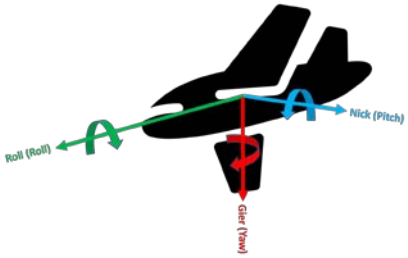
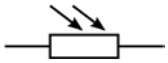
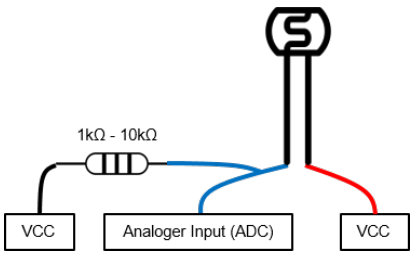
Tabelle 3 beschreibt die wichtigsten Sensoren.

Tabelle 3 - Eine Beschreibung der wichtigsten Sensoren

<table>
<tr><th>Potentiometer/verstellbarer Widerstand</th><th>Schaltung</th></tr>
<tr><td> 
Durch das Drehen des Reglers wird der Widerstand verändert, was eine proportional veränderte Spannung erzeugt (Ohmsches Gesetz). Wird bei Drehknöpfen oder für die Positionierung z.B. in der Robotik verwendet.
Eigenschaften:
✓ Analoger Input
✓ GPIO-Pin mit Analog-Digital-Wandler (ADC)
✓ Wertebereich von 0-1023 (bei 10 Bit ADC)</td><td> 
Mittlerer Pin ist das Signal.</td></tr>
<tr><th>Taste/Schalter/Knopf</th><th>Schaltung</th></tr>
<tr><td> 
Öffnet und schliesst den Stromkreis. Tastenzustand kann arretieren oder zurückspringen.

Eigenschaften:
✓ Digitaler Input
✓ Wertebereich entweder 0 oder 1
✓ Jeder GPIO Pin</td><td>Schaltung beim micro:bit/Calliope (mit bereits integriertem Pulldown-Widerstand) [5].
 
Schaltung mit Pull-down-Widerstand.
  </td></tr>
</table>

[5] Bei vielen Boards für die Schule ist der benötigte Pullup- oder Pulldown-Widerstand bereits eingebaut, darum kann dieser weggelassen werden. Ein Pullup- bzw. Pulldown-Widerstand gewährleistet, dass im offenen Zustand des Schalters keine unbestimmte Spannung herrscht, sondern dass der Pin auf VCC bzw. GND gezogen wird.

Lagesensor (Gyroskop/Beschleunigungssensor/Kompass)	**Schaltung**
  Ein Lagesensor (Inertial Measurement Unit) besteht aus einem Gyroskop, einem Beschleunigungssensor und einem Kompass. Dabei kann die Roll-Nick-Gier (Roll-Pitch-Yaw) Lage eines Systems berechnet werden. Jeder Sensor der Sensorengruppe liefert Informationen über die Lage. Aber erst die Kombination aller Sensoren liefert die beste Genauigkeit. Wird bei Navigationssystemen, Smartphones, Controllern für Gamekonsolen, Fitnessarmbändern usw. verwendet. **Gyroskop:** Misst Drehgeschwindigkeit auf 3 Achsen. **Beschleunigungssensor (Accelerometer):** Misst Beschleunigung auf drei Achsen. Im Ruhezustand zeigt der Accelerometer immer mindestens die Erdbeschleunigung (1g = 9.81m/s^2) an. **Kompass:** Misst die magnetische Ausrichtung. **Eigenschaften:** ✓ Digitaler oder analoger Input (je nach Sensor). ✓ Wertebereich z.B. 0-1023 bei 10 Bit ADC (oder bei bereits kalibriertem Sensor z.B: -180° bis +180°)	Der micro:bit/Calliope hat einen Lagesensor bestehend aus Gyroskop, Beschleunigungssensor und Kompass bereits integriert. Es muss also kein externer Sensor angehängt werden. Wenn man einen externen Lagesensor verwenden möchte, gibt es diese als digitale IMU-Sensoren. Das ist ein kleines Board mit integriertem Schaltkreis (IC), der die Sensorengruppe Gyroskop, Beschleunigungssensor und (meistens) Kompass enthält. Signale können je nach Sensor über analogen Input (ADC) mit je einem Pin für jede Achse oder über eine Kommunikationsschnittstelle (z.B. I^{2}C) übertragen werden.  
Lichtsensor	**Schaltung**
   Durch das Einfallen von Licht wird der Widerstand verändert, was eine proportional veränderte Spannung erzeugt (Ohmsches Gesetz). **Eigenschaften:** ✓ Analoger Input ✓ GPIO Pin mit Analog-Digital-Wandler (ADC) ✓ Wertebereich von 0-1023 (bei 10 Bit ADC)	Der micro:bit/Calliope hat einen Lichtsensor im LED-Display integriert. Für einen externen Lichtsensor eignet sich ein Fotowiderstand. 

2.3.2 Aktoren

Aktoren bewirken etwas in der Aussenwelt: sie wandeln elektrische Signale in physikalische Grössen um. Der Mikrocontroller steuert Aktoren, also sind Aktoren immer **Outputs**. Beispiele von Aktoren sind Motoren, LEDs, Lautsprecher usw. Auch bei den Aktoren wird zwischen analogen und digitalen Outputs unterschieden.

Analoger Output:
Einen einfachen Elektromotor, wie einen DC-Motor (z.B. Räder eines ferngesteuerten Autos) oder einen Vibrationsmotor (z.B. Vibrationsfunktion beim Smartphone), kann man stufenlos ansteuern: Ein DC-Motor oder Vibrationsmotor dreht bzw. vibriert umso schneller, je höher die Spannung ist, die an den Motor angelegt wird. Eine LED lässt sich ebenfalls durch das Verändern der Spannung kontinuierlich dimmen. Es wird also ein **analoges Signal in Form einer Spannung** benötigt. Da der Mikrocontroller in einer digitalen Welt lebt, kann er nicht so einfach ein analoges Signal an seinem Pin als Output erzeugen; der Pin kann jeweils nur entweder auf 0V oder 5V geschaltet werden. Um dennoch einen kontinuierlichen Output von 0V bis 5V zu erzeugen, wird ein kleiner Trick angewendet: die «Pulsweitenmodulation» (PWM). Der Pin wird dabei so schnell von 5V auf 0V hin und her geschaltet, dass eine Durchschnittsspannung am Pin erzeugt wird (Abbildung 19).
Auch die PWM-Funktion hat eine Auflösung, häufig beträgt diese 8 Bit. Das entspricht einem Wertebereich von 0 bis 255 (2^8-1). Das bedeutet, dass zwischen 0V und 5V unterschiedliche Spannungswerte in einer Feinheit von 0.0196 Volt erzeugt werden können (5V/255).

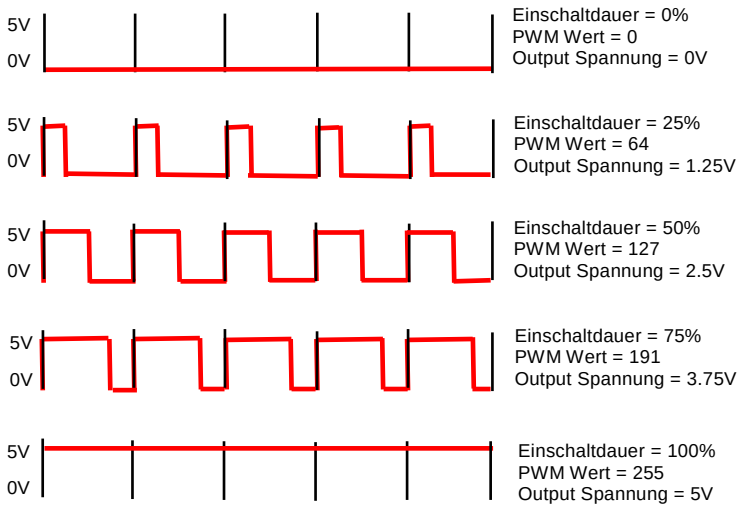


Abbildung 19 – Durch Pulsweitenmodulation (PWM) erreicht der Mikrocontroller eine kontinuierliche Durchschnittsspannung zwischen 0V-5V am Pin. PWM wird als analoger Output bezeichnet.

Zum Ausprobieren:
micro:bit/Calliope Challenge Nr. 6 – Ein Licht dimmen.
micro:bit/Calliope Challenge Nr. 7 – Einen Motor steuern.
Calliope Challenge Nr. 9 – Farben mit der Fingerspitze verändern.
micro:bit Challenge Nr. 14 – Die Farben des Regenbogens.

Digitaler Output:
Wie beim digitalen Input gibt es beim digitalen Output zwei Gruppen. Ein einfacher digitaler Output kann beispielsweise eine LED an- und ausschalten. Dabei kann sie nicht gedimmt werden, dafür wäre ein analoger Output notwendig. Wenn man einen DC Motor an einen digitalen Output anschliessen würde, wäre dieser entweder ausgeschaltet würde mit maximaler Geschwindigkeit drehen.

Die zweite Gruppe von digitalen Outputs sendet spezifische digitale Signale. Ein RC-Servo-Motor beispielsweise, welcher häufig im Modellbau und in der Robotik verwendet wird, benötigt ein bestimmtes digitales Signal um zu funktionieren. Ein RC-Servo-Motor hat eine Positionsregelung eingebaut, d.h., dass der Motor permanent seine Soll- und Ist-Position abgleicht und bei einer Abweichung den Fehler korrigiert. Aus diesem Grund werden RC-Servos dann verwendet, wenn eine präzise Position zwischen 0° und 180° benötigt wird (es gibt auch Servos, die eine volle Umdrehung machen können). Auch beim RC-Servo wird die Ansteuerung über Pulsweitenmodulation erzielt, sie funktioniert jedoch etwas anders als das PWM des oben beschriebenen analogen Outputs. Die genaue Positionierung eines RC-Servo-Motors kann dem Datenblatt des Herstellers entnommen werden. Üblich ist jedoch ein 50Hz-Signal von 20 Millisekunden Periodenlänge (1000ms/50). Alle 20ms wird ein Signal mit einem High-Pegel zwischen 1ms und 2ms erwartet. Ein High-Pegel von 1ms entspricht dabei der der Position des RC-Servos von 0°, 1.5ms von 90°, 1.75ms von 135°, 2ms von 180 (Abbildung 20). Es gibt noch weitere spezifische digitale Outputsignale. Ein Piezo-Buzzer benötigt beispielsweise einen digitalen Output in einem bestimmten Frequenzbereich, um die verschiedenen Pieptöne abspielen zu können. Tabelle 4 fasst die wichtigsten Aktoren zusammen.

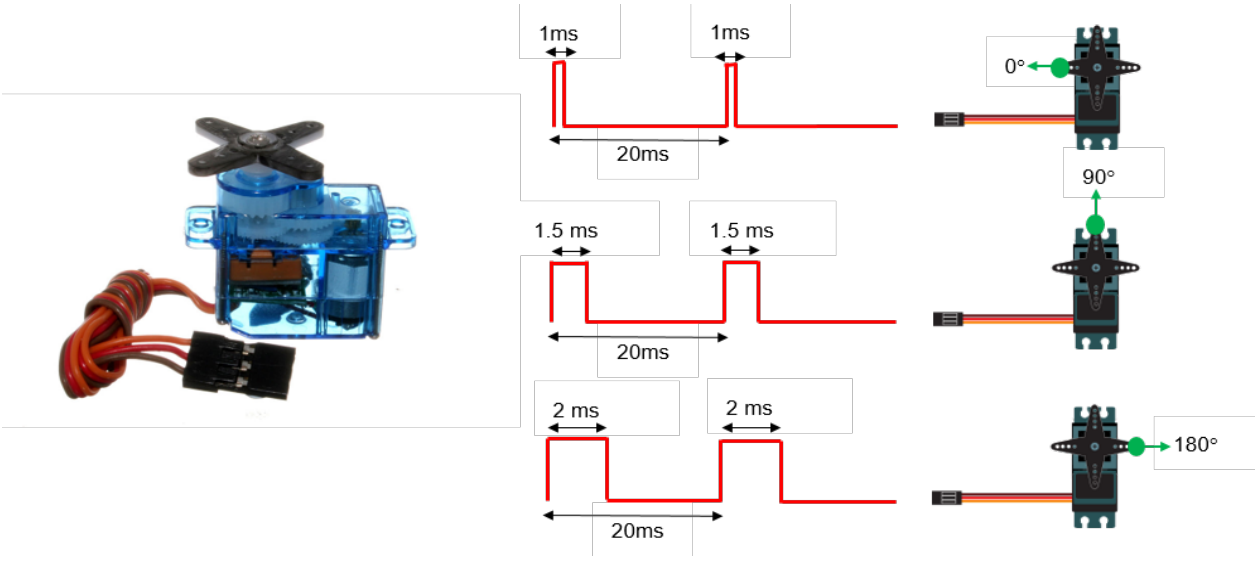


Abbildung 20 – Ein digitales Signal für die Steuerung eines RC-Servo Motors. Die Pulsweite bestimmt den Soll-Winkel des Motors.

Zum Ausprobieren:
micro:bit/Calliope Challenge Nr. 3 – Die Tasten A und B steuern das Licht.
micro:bit/Calliope Challenge Nr. 4 – Eine Taste steuert das Licht.
micro:bit/Calliope Challenge Nr. 8 – Musik komponieren und abspielen.
micro:bit Challenge Nr. 15 – Einen Servo-Motor steuern.
micro:bit Challenge Nr. 16 – Einen DC Motor steuern.

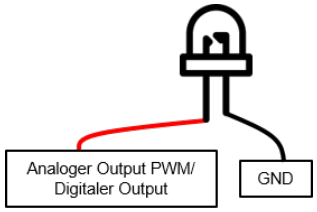
Tabelle 4 - Die wichtigsten Aktoren

<table>
<tr><th>LED (Light-Emitting Diode)</th><th>Schaltung</th></tr>
<tr><td>

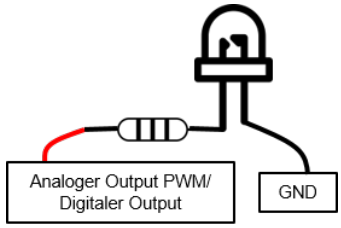
Licht in der Farbe gemäss LED.

Eigenschaften (Ein/Aus):
✓ Digitaler Output
✓ Wertebereich 0 oder 1
✓ Jeder GPIO-Pin

Eigenschaften (Dimmen):
✓ Analoger Output
✓ Wertebereich von 0-1023 (10 Bit) oder 0-255 (8 Bit)
✓ GPIO-Pin mit PWM Funktion
</td><td>
Schaltung mit bereits eingebautem Vorwiderstand (z.B. bei Calliope oder micto:bit).



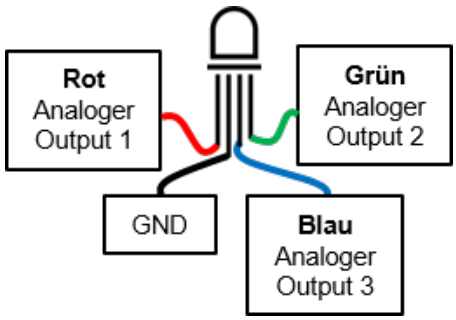
Auf Polarität achten!
Kürzeres Bein → GND

Schaltung ohne bereits eingebautem Vorwiderstand. Der Widerstand berechnet sich wie folgt:
$R = \frac{(U-UF)}{I}$


R= Widerstand, U=Versorgungsspannung, Vorwärtsspannung LED (Datenblatt der LED), I = Strom, der durch die LED fliessen soll (Datenblatt LED).
</td></tr>
<tr><th>RGB LED</th><th>Schaltung</th></tr>
<tr><td>

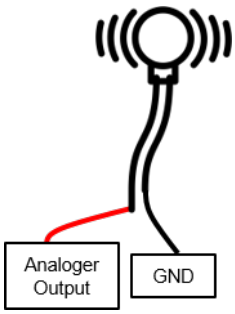
Licht im RGB-Farbraum (3 LEDs in einem: Rot, Grün, Blau).

Eigenschaften:
✓ Analoger Output
✓ Wertebereich von 0-1023 (10 Bit) oder 0-255 (8 Bit)
✓ 3 GPIO Pins mit PWM-Funktion
</td><td>
Auf Polarität achten!
Kürzestes Bein (Grün) → analoger Output, längstes Bein → GND, Bein neben Grün (Blau) → analoger Output, äusseres Bein (Rot) → analoger Output

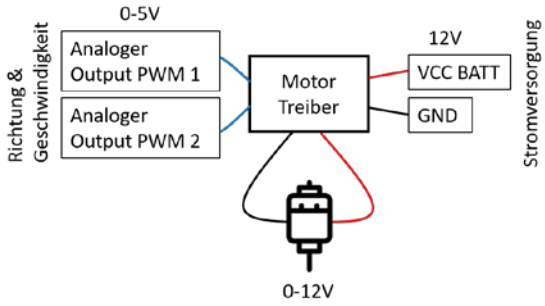
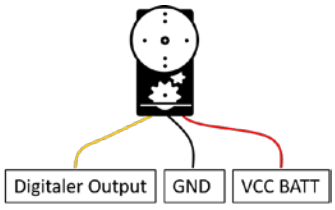
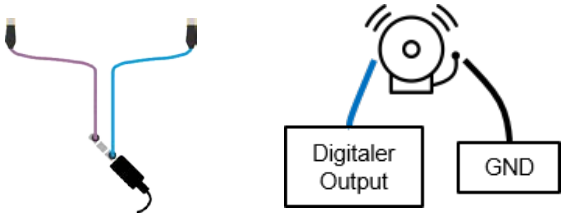
Ohne bereits eingebautem Widerstand muss je ein Vorwiderstand bei Rot, Grün und Blau benutzt werden.


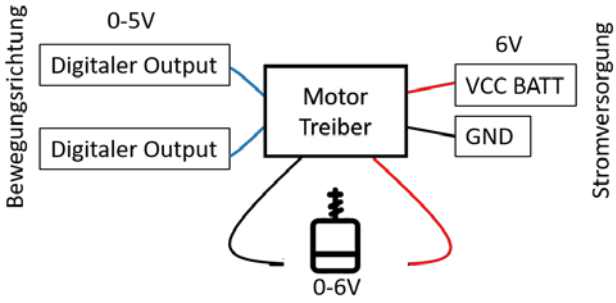
</td></tr>
<tr><th>Vibrationsmotor</th><th>Schaltung</th></tr>
<tr><td>


Vibriert mit unterschiedlicher Stärke.

Eigenschaften:
✓ Analoger Output
✓ Wertebereich von 0-1023 (10 Bit) oder 0-255 (8 Bit)
✓ GPIO-Pin mit PWM Funktion
</td><td>
Auf Polarität achten!
Das schwarze Kabel wird mit GND verbunden.
Ein kleiner Vibrationsmotor benötigt nicht so viel Strom, d.h. er kann direkt an einen micro:bit oder Calliope angeschlossen werden.


</td></tr>
</table>

DC-Motor (Gleichstrommotor)	Schaltung
 Der Gleichstrommotor dreht je nach Spannung mit unterschiedlicher Geschwindigkeit. **Eigenschaften:** ✓ Analoger Output ✓ Wertebereich von 0-1023 (10 Bit) oder 0-255 (8 Bit) ✓ 2 GPIO Pins mit PWM Funktion **Achtung! Muss immer an ein Motor-Treiber-Modul angeschlossen werden!** Der Calliope verfügt bereits über einen Motortreiber (über lötbare Pins). Für den micro:bit kann man einen Treiber hinzukaufen.	Wenn man einen DC-Motor direkt an eine Stromquelle schliesst, dann dreht er in eine Richtung. Wenn man die Polarität umdreht, dreht er in die andere Richtung. Die PWM-Funktion eines Mikrocontrollers kann die Polarität nicht umkehren und auch nicht genug Strom für den Motor liefern. Aus diesem Grund wird ein Motortreiber (H-Brücke) benötigt. Beispiel eines Motortreibers (es gibt unterschiedliche): 
RC-Servo-Motor	**Schaltung**
 Ein RC-Servo-Motor prüft immer seine Soll-Ist-Position und korrigiert diese. Dies ermöglicht eine präzise Positionierung. Wird z.B. im Modellbau oder in der Robotik verwendet. **Eigenschaften:** ✓ Digitaler Output ✓ Wertebereich in Winkelgrad 0°-180°	Es wird nicht empfohlen, den RC-Servo-Motor direkt an VCC des micro:bit oder Calliope zu schliessen. Ein Motor benötigt viel Strom und kann die Pins beschädigen. VCC BATT = VCC der Batterie direkt. **Auf Polarität achten!** Drei Kabel: Rot → VCC BATT, Braun/Schwarz → GND, Gelb/Orange → digitaler Output. Alle GND im System müssen immer verbunden werden. 
Piezo-Buzzer	**Schaltung**
 Verschiedene Piepstöne gemäss Frequenzbereich. **Eigenschaften:** ✓ Digitaler Output ✓ Wertebereich in Frequenzen Hz	Auf dem micro:bit funktioniert nur P0 für den Buzzer. **Polarität beachten!**  Wer keinen Buzzer zur Hand hat, kann auch einen Kopfhörer verwenden. Einfach an den Stecker clippen.

Linearer Motor (Solenoid)	Schaltung
 Im Gegensatz zur Drehbewegung der DC-Motoren und RC-Servo-Motoren bewegt sich der Stift dieses Motors linear vor- und zurück. Eigenschaften: ✓ Digitaler Output ✓ Wertebereich 0 oder 1 ✓ Jeder GPIO Pin **Achtung! Muss immer an ein Motor-Treiber-Modul angeschlossen werden!** Der Calliope verfügt bereits über einen Motortreiber (über lötbare Pins). Für den micro:bit kann man einen Treiber hinzukaufen.	Es wird nicht empfohlen, den Solenoid direkt an die Pins des micro:bit oder Calliope zu schliessen. Ein Motor benötigt viel Strom und kann sie beschädigen. Beispiel eines Motortreibers (es gibt unterschiedliche):  Die Bewegungsrichtung erfolgt über zwei digitale Outputs: 0/1 = eine Richtung, 0/0 = andere Richtung.

2.3.3 Stromversorgung

Die Mikrocontroller-Boards, die im Physical Computing verwendet werden, laufen meistens mit 3.3V oder 5V und benötigen niedrige Stromstärken im Milliampère-Bereich. Ein DC-Motor benötigt doppelt bis dreimal so viel Spannung. Durch die mechanische Arbeit, die Motoren verrichten, haben sie auch einen grösseren Strombedarf (bis zum Ampère-Bereich). Ein Pin eines Mikrocontrollers kann nur niedrige Stromstärken im Milliampère-Bereich liefern. Wenn nun ein Motor direkt am Pin angehängt wird, «zieht» dieser zu viel Strom. Die Leiterbahnen erhitzen sich und der Pin oder der ganze Mikrocontroller wird beschädigt.
Aus diesem Grund müssen Motoren an einen Motor-Treiber angeschlossen werden. Der Treiber gewährleistet, dass der Motor eine genügend hohe Spannung erhält und so viel Strom ziehen kann, wie er benötigt. Die PWM-Pins des Mikrocontrollers sind ebenfalls am Motor-Treiber angeschlossen. Diese steuern die gewünschte Geschwindigkeit und Bewegungsrichtung des Motors im 0V- bis 5V-Level. Der Motor-Treiber sorgt für das korrekte Mapping der Geschwindigkeit auf die 0V bis 12V des Motors.
Auch wenn sich die Differenz von 3.3V und 5V als geringfügig anhört, kann es durchaus sein, dass 1.7V zu viel einen Mikrocontroller oder Sensor beschädigen kann. Es muss immer beachtet werden, dass die Spannungen nicht überschritten werden. Beim Unterschreiten der Spannung, also wenn ein 5V-Board mit 3.3V versorgt wird, ist die korrekte Funktionsweise ebenfalls nicht gewährleistet.

VCC und GND dürfen auch niemals kurzgeschlossen, d.h. direkt verbunden werden. Aufgrund der hohen Stromstärken, die ein Kurzschluss generiert, kann auch bei niedrigen Spannungen ein Mikrocontroller oder Sensor leicht kaputtgehen. Die meisten Tools für die Schule beziehen den Strom über USB-Kabel, einer Serie von AA- oder AAA-Batterien (je 1.2V bei aufladbarer oder 1.5V bei herkömmlicher Batterie), oder Knopfzellenbatterien (bis 3.6V). Wenn man Batterien in Serie schaltet, addiert sich die Spannung. Bei Parallelschaltung addiert sich die Ladekapazität (Ampèrestunde). Diese Batterien sind für den Schulgebrauch unproblematisch. Bei einem Kurzschluss besteht keine grosse Verletzungsgefahr und viele Tools für die Schule sind robust. **Für die Schule wird nicht empfohlen, Lithium-Polymer-Akkus zu verwenden**, die gerne in der Robotik und im Modellbau wegen ihrer Ladekapazität verwendet werden. Bei einem Kursschluss setzen diese sehr viel Energie frei und können auch explodieren (siehe YouTube).

2.3.4 Wertebereiche von Inputs und Outputs

Der Datenfluss von der physischen Welt in die digitale und wieder zurück ist ein zentrales Thema im Physical Computing. Sensoren liefern Informationen in unterschiedlichen Formen, über digitale oder analoge Inputs. Ebenso benötigen Aktoren bestimmte analoge und digitale Signale. Bei der Programmierung muss man sich dabei immer überlegen, welche Arten von Informationen (Datentypen) vorliegen und wie sie weiterverarbeitet werden sollen, um das gewünschte Verhalten zu implementieren.

In den Kapiteln über analoge und digitale Inputs und Outputs wurden verschiedene Wertebereiche erwähnt. So liefert ein Potentiometer, das beispielsweise über den analogen Input am 10-Bit-Analog-Digital-Wandler angeschlossen ist, einen Wertebereich von 0 bis 1023. Ein integrierter Kompass einer Plattform kann bereits kalibrierte Messwerte von 0° bis 360° liefern, da die Software dies bereits so für den Nutzer bereitstellt. Ein digitaler Sensor über eine I²C Schnittstelle kann ebenso bereits aufbereitete Daten liefern im Gegensatz zu den «Rohdaten» des Analog-Digital-Wandlers. Ein DC-Motor benötigt über den PWM einen Wertebereich von 0 bis 255 und ein RC-Servo-Motor einen Winkel von 0° bis 180°. Ein Buzzer hat den Wertebereich in einer Frequenzspanne der Tonleiter.

Wenn man nun beispielsweise durch das Drehen des Reglers eines Potentiometers den Buzzer steuern möchte, müssen die unterschiedlichen Wertebereiche aufeinander abgestimmt werden. Dafür gibt es auf den meisten Physical-Computing-Plattformen eine bereitgestellte Funktion: **das Mapping**.

Abbildung 21 zeigt ein Beispiel eines Mapping in einer Blockprogrammiersprache. In der Anweisung «ring tone (Hz)» wird dem Buzzer über den roten Block eine Frequenz für einen bestimmten Ton übergeben. Dabei wird ein analoger Messwert X am Pin P1 gelesen (z.B. über ein Potentiometer), welcher einen Wertebereich von 0 bis 1023 hat, bzw. N1 bis H1. Der Messwert wird entsprechend in den Frequenzbereich der Tonleiter von 261Hz bis 1046Hz bzw. N2 bis H2 «gemappt». Mathematisch macht diese Map-Funktion Folgendes:

$$Frequenz\ Hz = (H2 - N2)\frac{(X - N1)}{(H1 - N1)} + N2$$

Bei einem Beispielmesswert von 345, wird ein Ton mit folgender Frequenz ausgegeben:

$$525.7\ Hz = (1046 - 261)\frac{(345 - 0)}{(1023 - 0)} + 261$$

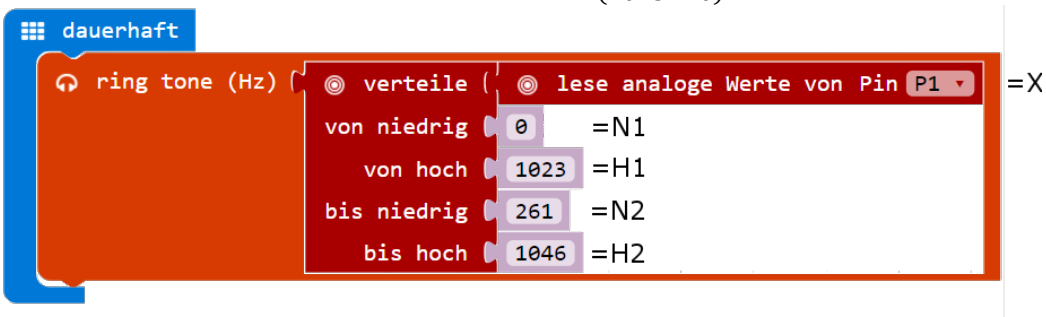


Abbildung 21 - Das Mapping eines analogen Inputs auf Pin P1 mit dem Wertebereich von 0-1023 zum Frequenzbereich eines Buzzers 261Hz - 1046Hz.

Zum Ausprobieren:
micro:bit Challenge Nr. 15 – Einen Servo-Motor steuern.

2.4 Programmierkonzepte

Die elektronischen Schaltungen mit Sensoren und Aktoren sind im Physical Computing nur ein Teil der Implementierung. Ohne ein passendes Programm für den Mikrocontroller passiert gar nichts. Für das Programmieren von Computern gibt es Programmierkonzepte, die auf jeder Plattform und über verschiedene Programmiersprachen hinweg allgemeingültig sind. Wer die Programmierkonzepte verstanden hat und sie für die eigene Implementierung anwenden kann, kann ohne viel Mühe auf andere Programmiersprachen und Hardwareplattformen umsteigen. Diese Kompetenz ist im Lehrplan 21 aus diesem Grund auch explizit aufgeführt (Kapitel 1.5). In den folgenden Kapiteln werden die Programmierkonzepte kurz beschrieben. Es ist möglich, dass sie für Einsteigerinnen und Einsteiger zunächst noch etwas abstrakt wirken. Mit zunehmender Erfahrung wendet man diese Konzepte dann mit der Zeit automatisch an.

Die Beispiele sind mittels Blockprogrammierung dargestellt, da sie besser lesbar ist als eine textuelle Darstellung des Programmcodes. Zudem entspricht sie der Programmiersprache der micro:bit- und Calliope-Challenge-Karten, die Teil dieses Hefts bilden. Es spielt keine Rolle, wie der Programmcode dargestellt wird, die Programmierkonzepte bleiben immer dieselben.

2.4.1 Anweisung

Ein Computerprogramm besteht aus einer Vielzahl von Anweisungen. Eine Anweisung ist ein eindeutiger Befehl, der ein Objekt im Programm manipuliert. Ein Objekt kann dabei rein virtuell im Programm existieren, wie beispielsweise eine Variable. Das Objekt kann aber auch eine physische Repräsentation als Input oder Output auf der Plattform haben, wie z.B. Pins, Sensoren oder Aktoren.

Das Erzeugen einer Spannung an einem Pin, die Ausgabe von Musik, die Zuordnung eines Werts einer Variablen usw. wird alles über Anweisungen durchgeführt. Anweisungsblöcke haben in blockbasierten, grafischen Programmierumgebungen oftmals diese Form (Abbildung 22):

Abbildung 22 – Ein Anweisungsblock.

Eine Programmierumgebung stellt bereits vordefinierte Anweisungen zur Verfügung. Die Gesamtheit der zur Verfügung gestellten Anweisungen ist die Bibliothek oder «Library». Die meisten Programmierumgebungen bieten eine Library für die grundsätzlichen Anweisungen zur Ein- und Ausgabe, sowie zur Verarbeitung zur Verfügung. Anweisungsblöcke, die dieselben Objekte manipulieren, sind in blockbasierten Programmierumgebungen oftmals in der gleichen Farbe dargestellt und in einer Befehlsgruppe im Menü zusammengefasst.

Die Anweisung in Abbildung 23 greift auf ein physisches LED-Display zu und zeigt den Lauftext «Hello!» an.

Abbildung 23 – Anweisung für die Anzeige eines Lauftexts auf einem LED-Display.

Die Anweisung in Abbildung 24 greift auf den physischen Buzzer zu und spielt eine Melodie einmal.

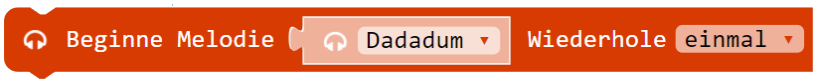


Abbildung 24 – Anweisung für das Spielen einer Melodie auf einem Buzzer

Die Anweisung in Abbildung 25 deklariert eine Variable mit dem Namen «zähler» und dem Wert 3.

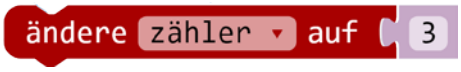


Abbildung 25 – Anweisung für die Deklaration einer Variablen und der Zuordnung eines Werts.

Mittels Unterprogrammen kann man eigene Anweisungen schreiben und die Standard-Library der Programmierumgebung ergänzen. Mehr dazu siehe Kapitel 2.4.10.

2.4.2 Sequenz

Eine Sequenz ist eine Abfolge von Anweisungen, die untereinander zusammengesteckt werden können. Sie werden nacheinander (sequentiell) von oben nach unten ausgeführt.

Im folgenden Beispiel werden drei verschiedene Symbole nacheinander auf dem LED-Display angezeigt, zuerst das Herz, dann das Kreuz und zuletzt das Quadrat (Abbildung 26).

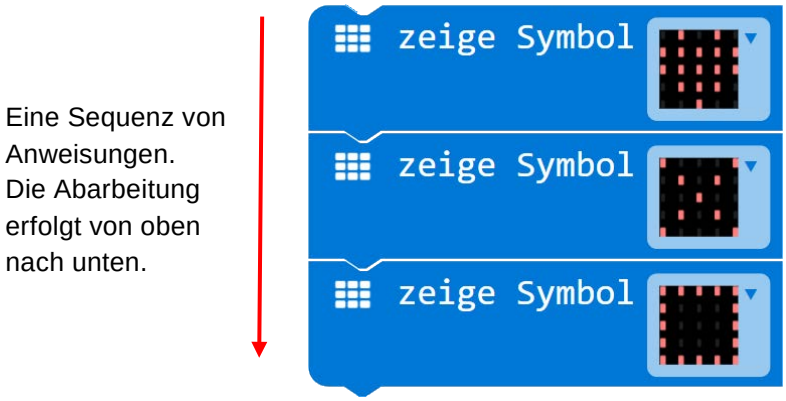


Abbildung 26 – Eine Sequenz von Anweisungen.

Passende Blöcke schnappen ineinander. So können beliebig viele verschiedene Anweisungen zusammengesteckt werden (Abbildung 27).

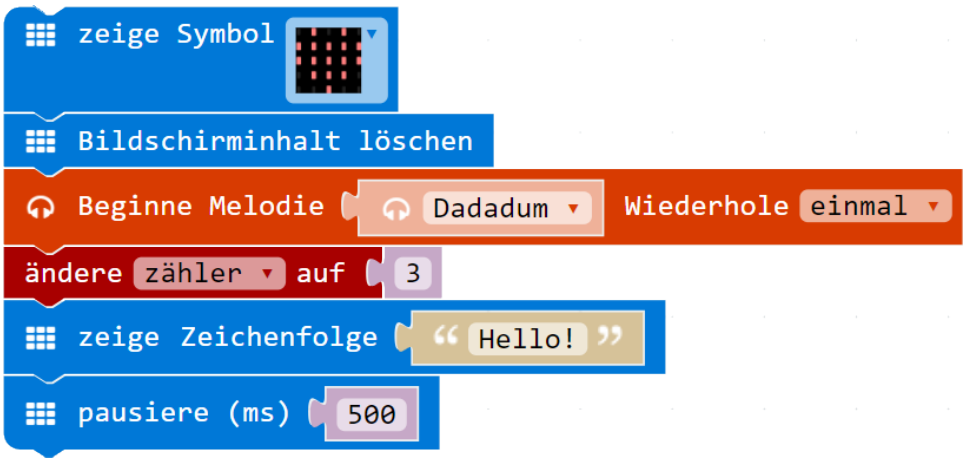


Abbildung 27 – Beliebig viele verschiedene Anweisungen können zusammengesteckt werden.

2.4.3 Programmeinstieg und Endlosschleife

Jedes Computerprogramm hat einen Startpunkt, an dem die ersten Anweisungen durchgeführt werden. Physical-Computing-Plattformen starten oftmals automatisch das Programm beim Anschliessen der Stromversorgung oder nach einem «Reset» über die Reset-Taste. Die Startfunktion ist mit dem «beim Start»-Block dargestellt und steht als eigener Block im Programmcode. Die darin enthaltenen Anweisungen dienen oft zur Initialisierung, d.h. es werden wichtige Variablen definiert, die im Verlauf des Programms benötigt werden, Sensoren werden kalibriert, eine Kommunikationsschnittstelle wird aufgebaut usw. Nachdem die Anweisungen durchgeführt wurden, springt der Programmverlauf in die «dauerhaft- Schleife, wenn eine solche existiert. Wenn nicht, dann wird das Programm beendet. Im Beispiel von Abbildung 28 wird beim Start ein Herz auf dem LED-Display angezeigt. Es kann nur ein einziger «beim Start»-Block im Programmcode existieren.

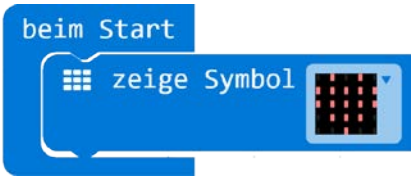


Abbildung 28 – Programmeinstiegsblock

Beim interaktiven Systemen muss ein Programm ständig laufen, da auf Signale von Sensoren reagiert werden soll. Die «dauerhaft» oder Endlosschleife macht genau das und steht ebenfalls als eigener Block im Programmcode (Abbildung 29). Die Sequenz innerhalb dieser Schleife wird ständig wiederholt. Nach der letzten Anweisung springt der Programmverlauf wieder zur obersten Anweisung. Im folgenden Beispiel werden die Symbole Herz, Kreuz, Quadrat, Herz, Kreuz, Quadrat, Herz usw. nacheinander ununterbrochen auf dem LED-Display angezeigt.

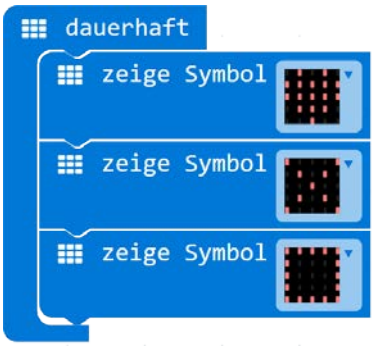


Abbildung 29 - Eine Endlosschleife

Einige Programmierumgebungen lassen es zu, mehrere Endlosschleifen im Programmcode nebeneinander zu haben. Die Schleifen werden dann pseudo-parallel durchgeführt, d.h. sie werden zwar alle durchgeführt, aber dennoch sequentiell über die Schleifen hinweg. In welcher Abfolge dies geschieht, entscheidet die jeweilige Programmierumgebung.
Ein Programmcode muss mindestens einen «beim Start»-Block oder eine «dauerhaft»-Schleife oder beides zusammen enthalten.

2.4.4 Parameter

Parameter liefern Informationen über ein Objekt im Programm. Dabei kann das Objekt virtuell existieren (z.B. das Ergebnis eines mathematischen Ausdrucks, eine Variable) oder eine physische Repräsentation haben (z.B. Pin, Sensor, Aktor). Parameter haben in blockbasierten Programmierumgebungen oftmals diese Form (Abbildung 30):

Abbildung 30 – Ein Parameterblock

Die Information, die in Parametern gespeichert ist, kann verschiedene Formen haben (Abbildung 31): (a) einen Zahlenwert «0», (b) eine Zeichenkette «Hello!», (c) einen Zustand «wahr», (d) den Wert einer Variablen, (e) das Ergebnis einer Rechnung, (f) eine Zufallszahl, (g) ein Pixelbild. Diese unterschiedlichen Formen der Information, auch «Datentypen» genannt, müssen zu der Anweisung passen, in die sie eingefügt werden[6]. So lässt eine Blockprogrammiersprache erst gar nicht zu, unpassende Parameter mit einer Anweisung zu verknüpfen. Eine Zeichenkette kann nicht der Anweisung «zeige Nummer» hinzugefügt werden. Dafür wird die Anweisung «zeige Zeichenfolge» verwendet.

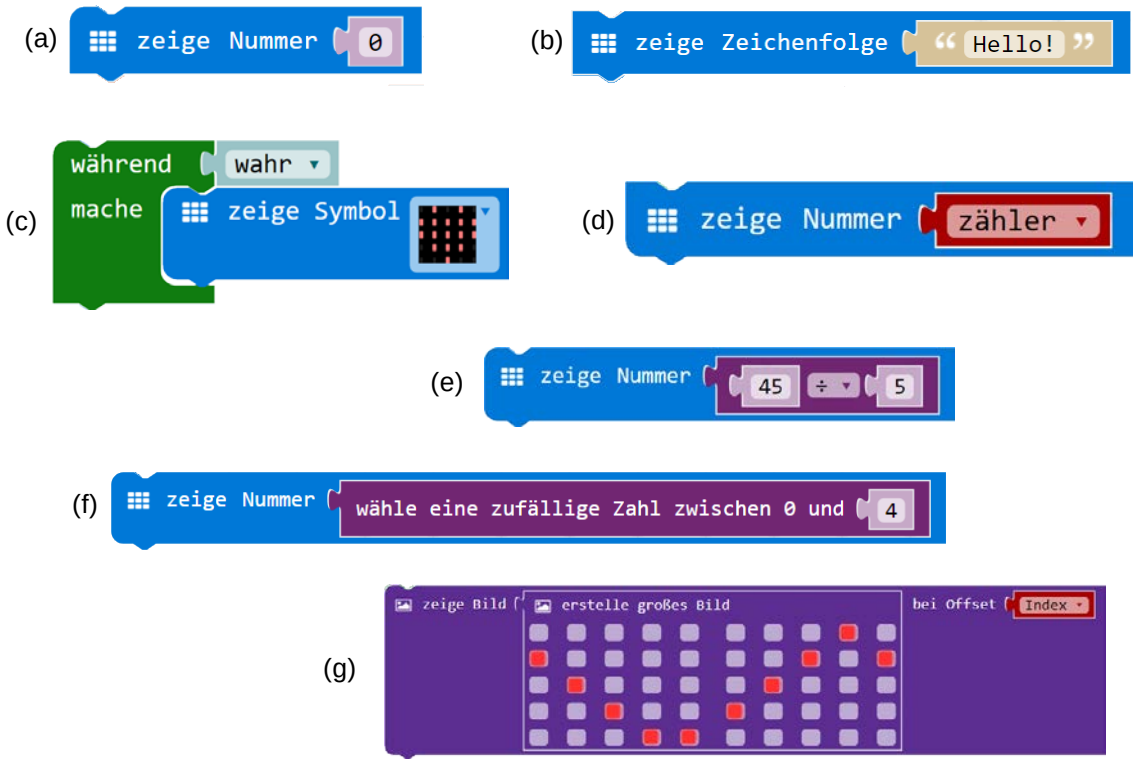


Abbildung 31 – Verschiedene Arten von Parametern. (a) Eine Zahl, (b) eine Zeichenkette, (c) ein Zustand «wahr», (d) der Wert einer Variablen, (e) das Ergebnis einer Rechnung, (f) eine Zufallszahl, (g) ein Pixelbild.

Parameter lassen sich zudem beliebig verschachteln. In Abbildung 32 sind sieben Parameterblöcke ineinander verschachtelt, um (40+5)/(1x5) zu berechnen. Dabei wird von innen nach aussen berechnet (wie bei Klammern in der Mathematik).

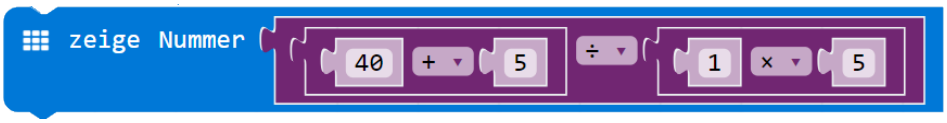


Abbildung 32 – Parameter kann man beliebig verschachteln.

[6] In anderen Programmiersprachen wird ein Parameter, der einer Anweisung übergeben wird, auch als «Argument» bezeichnet. Diese Bezeichnung kann als Synonym betrachtet werden.

Sensoren liefern Informationen. Aus diesem Grund werden diese in der Blockprogrammierung ebenfalls über Parameter ausgelesen. Abbildung 33(a) zeigt eine Anweisung, um den digitalen Wert «1» auf den Pin P0 zu schreiben. In der verschachtelten Version (b) wird der digitale Wert von Pin P1 zunächst ausgelesen und dann der Anweisung «schreibe digitalen Wert von Pin P0» übergeben. So kann direkt der digitale Input P1 mit dem digitalen Output P0 verknüpft werden.

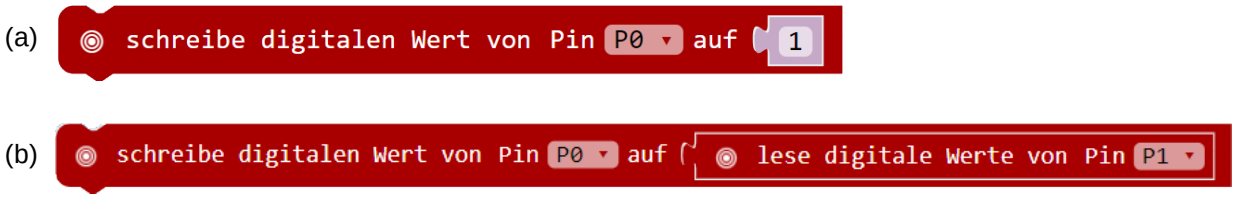


Abbildung 33 - Über Parameter werden Sensoren ausgelesen.

2.4.5 Schleife

In einem Programmcode reihen sich Anweisungen in langen Sequenzen aneinander. Um den Programmcode kompakter zu gestalten, werden sich wiederholende Anweisungssequenzen in Schleifen verpackt. Durch Mustererkennung identifiziert man Programmcode, der sich wiederholt. Mehr zur Mustererkennung im Kapitel 2.5.

Es werden drei verschiedene Arten von Schleifen unterschieden:

Wiederhole x-mal (FOR):
Die Schleife in Abbildung 34(a) wiederholt die Sequenz innerhalb der Schleife genau so oft, wie in dem Parameter eingegeben. Der Programmcode spielt sechs Noten CDDCDD hintereinander.

Die Schleife in Abbildung 34(b) wiederholt ebenfalls den Programmcode so oft, wie es der Wert des Parameters festlegt. Der einzige Unterschied zum Beispiel (a) ist der Index. Dies ist eine Variable, welche die Zahl des aktuellen Durchlaufs enthält. Diese fängt bei 0 an und endet bei 4. Der Code «zeige Nummer» wird also fünfmal durchlaufen[7]. Die Nummern 0, 1, 2, 3, 4 werden nacheinander auf dem LED-Display angezeigt.

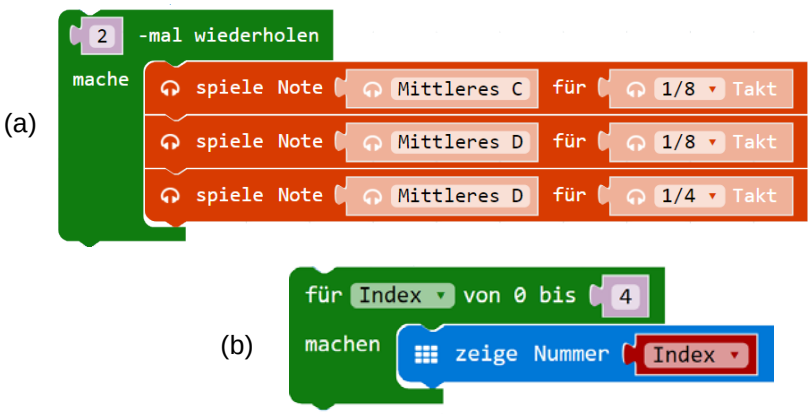


Abbildung 34 – Eine Wiederhole x-mal Schleife (FOR). (a) Der Parameter gibt an, wie oft die Schleife wiederholt wird. (b) der Index enthält die aktuelle Zahl der Wiederholung.

[7] In der Informatik fängt der Zähler immer bei 0 an, was demzufolge der erste Durchlauf ist.

Wiederhole fortlaufend (FOREVER):
Die Sequenz innerhalb der Schleife wird unendlich oft wiederholt (siehe Kapitel 2.4.3.).

Wiederhole solange (WHILE):
Diese Schleife prüft zu Beginn eines jeden Durchlaufs, ob der Wert des Parameters «wahr» ist. Nur dann werden die Anweisungen darin ausgeführt. Die Zeichenfolge «Hello!» wird im Beispiel von Abbildung 35(a) unendlich oft wiederholt. Diese Schleife mit dem konstanten Parameter «wahr» macht dasselbe wie der «dauerhaft»-Block.

Ändert man nun den Parameter auf «falsch», wird die Schleife nie ausgeführt (Abbildung 35(b)). Durch die hier etwas irreführende Darstellung könnte der Eindruck entstehen, dass die Schleife dann ausgeführt wird, solange der Parameter «falsch» ist. Dies ist aber nicht der Fall. «Falsch» bricht die Schleife immer ab bzw. führt sie gar nie aus.

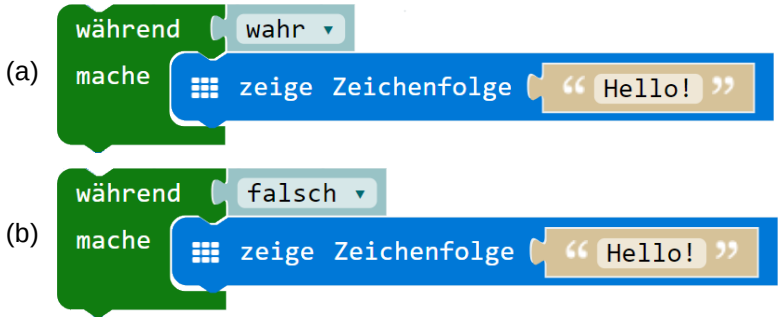


Abbildung 35 – Eine «wiederhole solange» Schleife (WHILE). (a) wenn der Parameter «wahr» ist, wird die Schleife ausgeführt. (b) bei «falsch» wird die Schleife abgebrochen.

Die «wiederhole solange» Schleife macht vor allem Sinn, wenn sich der Parameter «wahr» oder «falsch» während der Laufzeit des Programms ändert. Bleibt der Parameter konstant wie bei den Beispielen oben, wird die Schleife nie verlassen oder gar nie ausgeführt. Im folgenden Beispiel kann der Parameter den Wert «wahr» oder «falsch» enthalten. Solange die Taste A gedrückt wird, ist der Parameter «wahr» und die Schleife wird wiederholt. Sobald die Taste losgelassen wird, ist der Parameter «falsch» und die Schleife wird verlassen (Abbildung 36(a)). Dies wird auch «Abbruchbedingung» genannt.

Um den Fall umzukehren, d.h. «wiederhole bis Taste A gedrückt wird», muss ein «nicht» Parameter verwendet werden. Denn das «wahr» von «Button A ist gedrückt», wird durch die Negation zu einem «nicht wahr» und dazu zu einem «falsch». Dies ist die Abbruchbedingung und die Schleife wird verlassen (Abbildung 36(b)). Mehr zu logischen Verknüpfungen im Teil Boolesche Algebra 2.4.9.

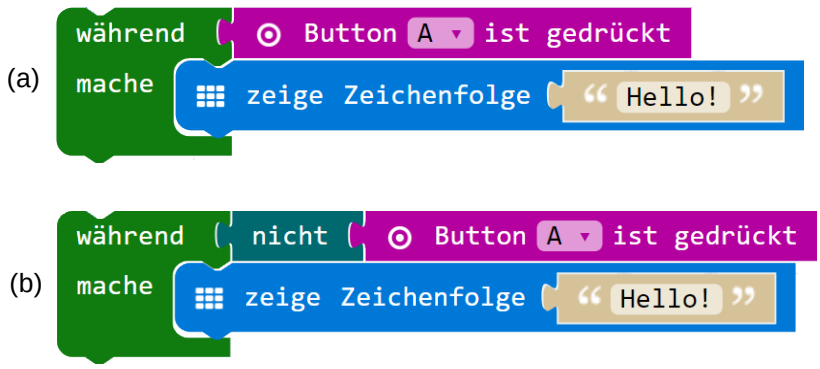


Abbildung 36 – Die Abbruchbedingung ist abhängig von der Taste A. (a) Wiederhole, solange Taste A gedrückt ist. (b) Wiederhole, bis Taste A gedrückt ist.

2.4.6 Bedingung

Um ein Programm interaktiv zu gestalten, muss die Ausführung einer Anweisung oder einer Sequenz von bestimmten Faktoren während der Programmlaufzeit abhängig gemacht werden. Dazu eignen sich Bedingungsblöcke.

Eine Bedingung prüft, ob eine Frage mit «wahr» beantwortet werden kann. Falls ja, dann wird die Anweisung oder die Sequenz innerhalb des Bedingungsblocks neben «dann» ausgeführt (Abbildung 37(a)). Wenn die Antwort auf die Frage «falsch» ist, dann wird der Programmcode übersprungen. Unmittelbar nach dem Bedingungsblock wird das Programm wieder fortgesetzt. Ändert man nun den Parameter auf «falsch», wird die Anweisung unter «dann» übersprungen (Abbildung 37(b)). Durch die hier etwas irreführende Darstellung könnte der Eindruck entstehen, dass die Bedingung dann ausgeführt wird, wenn der Parameter «falsch» ist. Dies ist aber nicht der Fall. «Falsch» führt die Anweisung nie aus.

Neben «Wenn/Dann» (IF/ELSE), gibt es noch die «Wenn/Dann/Sonst» (IF/ELSEIF/ELSE) Bedingung (Abbildung 37(c)). Der einzige Unterschied ist, dass hier bei einer nicht wahren Antwort die Anweisungen im Teil «ansonsten» ausgeführt werden. In diesem Block wird folglich immer entweder der «dann»-Teil oder der «ansonsten»-Teil ausgeführt.

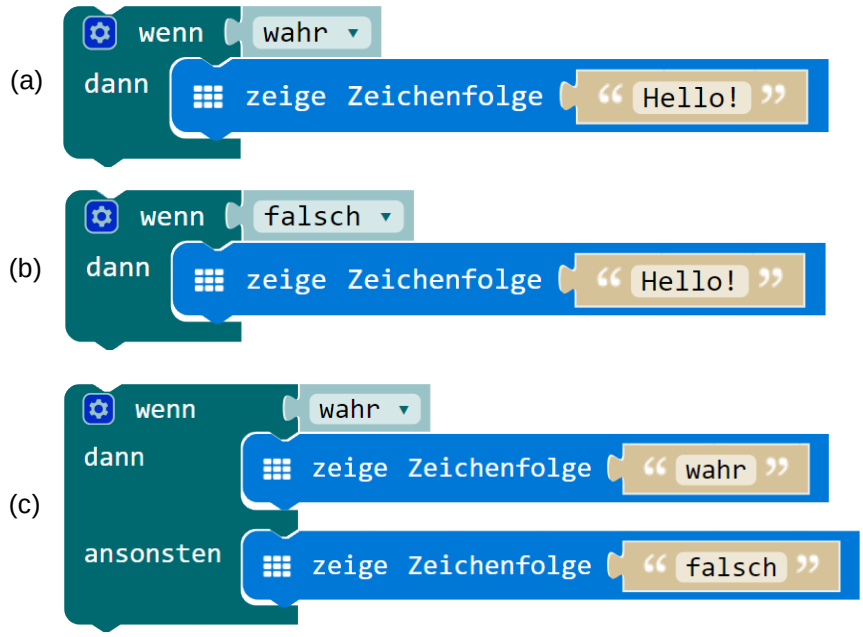


Abbildung 37 – Bedingungsblock IF/ELSE und IF/ELSEIF/ELSE. (a) Wenn die Bedingung «wahr» ist, wird die Anweisung ausgeführt. (b) Wenn die Bedingung «falsch» ist, wird die Anweisung nicht ausgeführt. (c) Wenn «wahr», dann wird die erste Anweisung ausgeführt und wenn «falsch» dann die zweite.

Eine Bedingung macht dann vor allem Sinn, wenn sich der Parameter während der Programmlaufzeit verändert und somit äussere Faktoren den Programmfluss bestimmen. Bedingungen können beliebig mit weiteren Bedingungspaaren «wenn/dann» erweitert werden. Im folgenden Beispiel wird zuerst geprüft, ob die beiden Tasten A und B gleichzeitig gedrückt sind (Abbildung 38). Wenn ja, dann wird die Zeichenfolge «A+B» auf dem LED-Display ausgegeben und der ganze Bedingungsblock verlassen. Wenn die erste Be-

dingung nicht zutrifft, wird die zweite Bedingung «ist Taste A gedrückt» geprüft. Wenn ja, wird die entsprechende Anweisung ausgeführt und der ganze Bedingungsblock verlassen. Wenn nein, wird die nächste Bedingung geprüft usw. Wenn gar keine Bedingung zutrifft, wird schliesslich die Anweisung unter «ansonsten» ausgeführt.

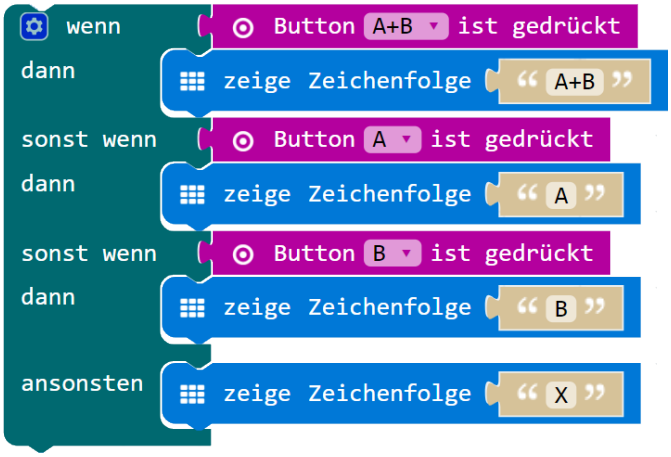


Abbildung 38 – Bedingungsblöcke können beliebig erweitert werden.

Es ist dabei zu beachten, dass in einem Bedingungsblock die Bedingungen immer strikt von oben nach unten geprüft werden. Das folgende Beispiel unterscheidet sich vom oberen nur durch die Anordnung der Bedingungen (Abbildung 39). Dadurch, dass die Bedingung «sind Tasten A+B gedrückt» nun als letztes geprüft wird, kommt es gar nie dazu. Denn wenn beide Tasten A und B gedrückt sind, ist die erste Bedingung «ist Taste A gedrückt» immer erfüllt und die entsprechende Anweisung wird ausgeführt und danach der ganze Block verlassen.

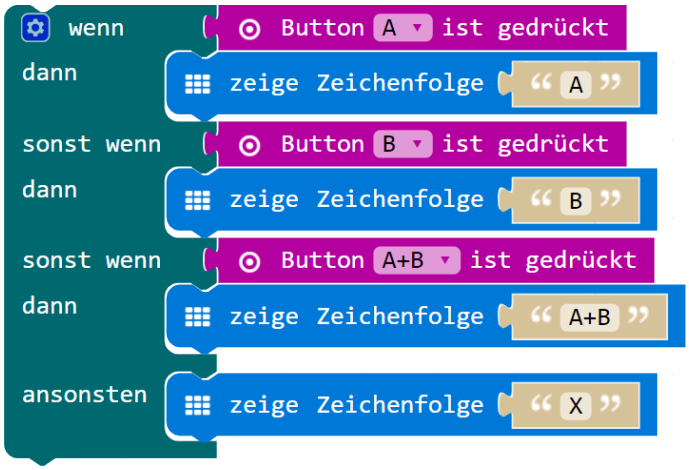


Abbildung 39 – In diesem Bedingungsblock kommt es nie zur Frage, ob die Tasten A+B gedrückt sind, da die erste Bedingung in diesem Fall immer erfüllt ist.

Wenn man sicherstellen möchte, dass immer alle Bedingungen geprüft werden, muss man diese aus einzelnen Bedingungsblöcken zusammenstellen (im Gegensatz zum kombinierten Bedingungsblock von Abbildung 38). Im folgenden Beispiel werden nun alle Bedingungen nacheinander geprüft (Abbildung 40). Im Fall, dass beide Tasten A und B gedrückt sind, werden jedoch auch alle drei Anweisungen ausgeführt. Ein Fall «ansonsten» funktioniert bei dieser Variante nicht.

Abbildung 40 - Einzelne Bedingungen untereinander werden immer geprüft. «Ansonsten» funktioniert hier jedoch nicht.

2.4.7 Ereignis (Interrupt)

Ein weiteres wichtiges Programmierkonzept, um ein Programm interaktiv zu gestalten, ist das Ereignis. Sobald ein Ereignis eintrifft, wird die Sequenz innerhalb des Ereignisblocks sofort ausgeführt. Ereignisse stehen für sich alleine im Programmcode. Ereignisse haben in der Blockprogrammierung oftmals diese Blockform (Abbildung 41):

Abbildung 41 – Ein Ereignisblock.

Egal, an welcher Position sich das Programm innerhalb des «dauerhaft» Blocks gerade zum Zeitpunkt des Eintreffens des Ereignisses befindet, der Ereignisblock wird immer sofort ausgeführt. Dieses Verhalten wird auch als «Interrupt» bezeichnet. Nach der Abarbeitung der Anweisungen oder Sequenzen innerhalb des Ereignisblocks wird dieser wieder verlassen und der ursprüngliche Programmverlauf fortgesetzt. Im folgenden Beispiel in Abbildung 42 muss die Anweisung «zeige Zeichenfolge X» im «dauerhaft» Block untergebracht werden, da es kein Ereignis für «keine Tasten gedrückt» gibt (dies würde auch keinen Sinn machen). Nach dem Ereignis springt das Programm so schnell wieder in den «dauerhaft» Block zurück, dass man die Zeichenfolge des Ereignisblocks auf dem LED-Display kaum wahrnimmt. Aus diesem Grund wurde eine Pause von 2 Sekunden (2000 Millisekunden) innerhalb des «dauerhaft»-Blocks hinzugefügt.

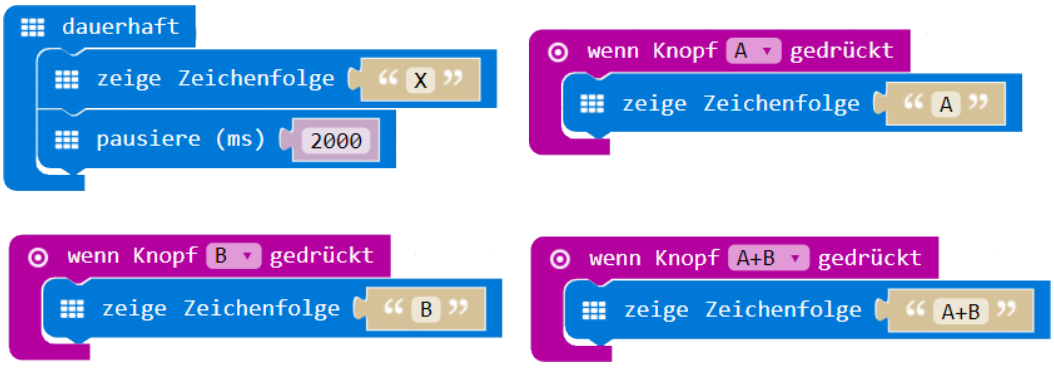


Abbildung 42 - Ein Programmcode bestehend aus drei Ereignisblöcken und einer Endlosschleife.

Ereignis (Interrupt) vs. ständige Abfrage (Polling):
Im Physical Computing ist die Abfrage von Sensoren ein zentraler Bestandteil um interaktive Systeme zu bauen. Es ist möglich, das selbe Verhalten eines Systems mit unterschiedlichem Programmcode zu erzielen, d.h. es gibt immer mehrere korrekte Lösungen. Der folgende Programmcode ist ein Beispiel dafür, wie man die Sensoren ständig abfragt, um den Fall eines Tastendrucks zu entdecken (Abbildung 43). Dieses ständige Abfragen wird auch als «Polling» bezeichnet. Der ereignisbasierte Code über «Interrupts» im Beispiel von Abbildung 42 erzielt schlussendlich dasselbe Verhalten.

Welches Programmierkonzept eignet sich nun für die Implementierung dieses Verhaltens: Polling oder Interrupt? Die Frage kann nicht pauschal für jedes Problem beantwortet werden. Polling wird in der Hauptschleife «dauerhaft» implementiert, d.h. es ist gut möglich, dass unter- oder oberhalb des Programmcodes für das Abfragen der Sensoren noch viele andere Anweisungen stehen, etwa Anweisungen für die Ausgabe von Melodien, LED-Display Lauftext, Berechnungen, Würfeln von Zufallszahlen, Pausen, Kommunikation usw. Die Abarbeitung all dieser anderen Anweisungen in der «dauerhaft»-Schleife benötigt Zeit und die Abfrage der Sensoren erfolgt erst, wenn die Ausführung des Programms bei den Bedingungen angekommen ist. Es kann also sein, dass der Tastendruck nicht ebenso schnell detektiert wird wie bei der Implementierung über Ereignisse. Wenn man viele verschiedene Sensoren abfragt, macht die Variante über Ereignisse mehr Sinn, da sie einerseits übersichtlicher ist und andererseits die «dauerhaft»-Schleife nicht mit ständigen Abfragen beschäftigt wird.

Wenn es ein Ereignis für das Lösen des Problems gibt, dann sollte man es zunächst über diese Variante versuchen. Für viele Situationen (z.B. bei analogen Sensoren «wenn Helligkeit dunkel», «wenn Temperatur tief» usw.) gibt es jedoch keine passenden vordefinierten Ereignisblöcke. Adiesem Grund muss man die Lösung über Polling implementieren.

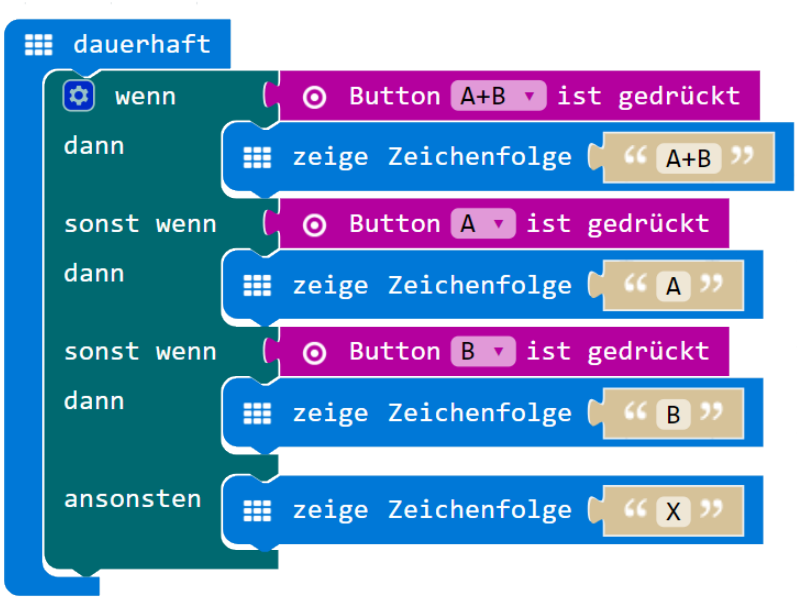


Abbildung 43 – Über «Polling» in der Endlosschleife kann man die Sensoren ständig abfragen.

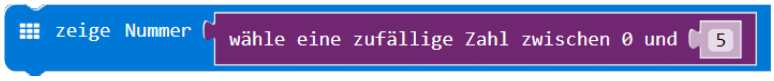
2.4.8 Variable

Oftmals möchte man gewisse Werte speichern und an verschiedenen Orten im Programm wiederverwenden. Dafür eignen sich Variablen (manchmal auch «Platzhalter» genannt). In der Blockprogrammierung haben Variablen zwei unterschiedliche Blöcke (Abbildung 44): den Definitionsblock (links) und den dazugehörigen Parameter (rechts). Über den Definitionsblock wird die Variable deklariert und ihr gleichzeitig einen Wert zugewiesen. Dieser bleibt solange bestehen, bis der Wert über dieselbe Anweisung später im Programm wieder überschrieben wird. Der dazugehörige Parameterblock liefert den aktuellen Wert der Variablen.

Abbildung 44 – Der Definitionsblock und dazugehörige Parameter einer Variablen.

In Abbildung 45(a) wird eine Zufallszahl gewürfelt und auf dem LED-Display angezeigt. Das Würfeln geschieht innerhalb des Parameterblocks. Sobald der Parameter den Wert an die Anweisung «zeige Nummer» übergeben hat, ist dieser «verloren», d.h. man hat keinen Zugriff mehr auf die Zufallszahl. In Abbildung 45(b) wird die Zufallszahl in einer Variablen mit dem Namen «zufallszahl» gespeichert. Über den dazugehörigen Parameter kann auf diese Zahl auch später im Programm zugegriffen werden.

(a)



(b)

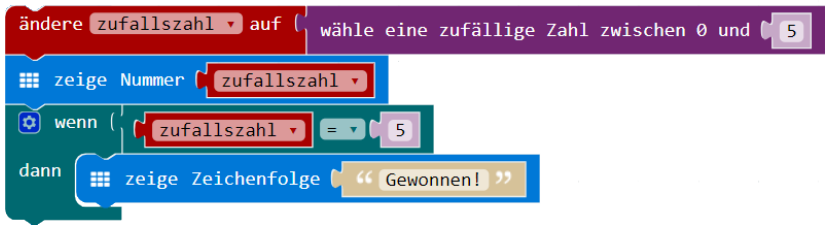
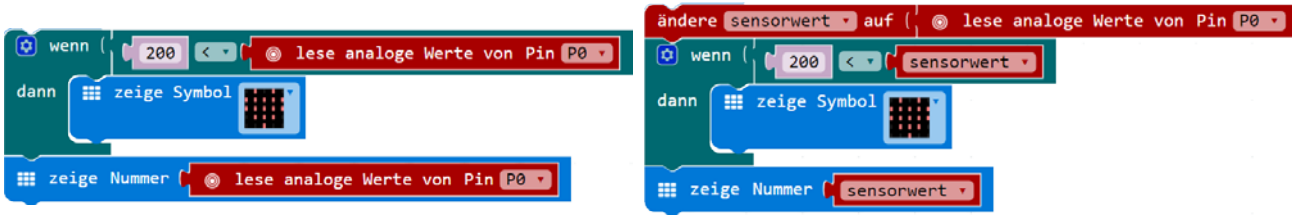


Abbildung 45 – Variablen speichern Werte für den späteren Gebrauch im Programm. (a) Nach der Anzeige auf dem LED-Display hat man keinen Zugriff mehr auf die Zufallszahl. (b) Eine Variable speichert den Wert der Zufallszahl, damit sie später wiederverwendet werden kann.

Oftmals ist es vor allem beim Lesen von Messwerten über Sensoren sinnvoll, den Wert in einer Variablen zu speichern. In Abbildung 46(a) wird der Messwert zweimal ausgelesen. Auch wenn es sich um wenige Millisekunden handelt, kann sich dieser bereits geändert haben. (b) Die saubere Implementierung über eine Variable. Hier kann man davon ausgehen, dass derselbe Messwert verarbeitet wird.



(a) (b)

Abbildung 46 – Speichern eines Messwerts über eine Variable. (a) Hier wird der Messwert zweimal ausgelesen und kann unterschiedlich sein. (b) Über die Variable kann man davon ausgehen, dass derselbe Messwert weiterverarbeitet wird.

2.4.9 Boolesche Algebra (Aussagenlogik)

Eines der wichtigsten Werkzeuge beim Programmieren ist die Boolesche Algebra. Sie besteht aus Ausdrücken, die entweder «wahr» (TRUE) oder «falsch» (FALSE) sind. Die Grundkomponenten der Booleschen Algebra werden in der Blockprogrammierung über Parameterblöcke dargestellt (Abbildung 47).

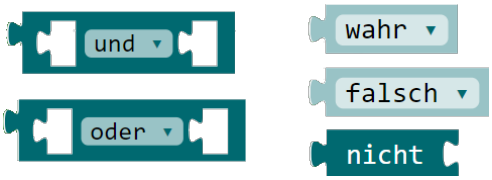


Abbildung 47 – Ausdrücke der Booleschen Algebra.

UND (AND):
Bei einer UND-Verknüpfung müssen immer alle Teilaussagen «wahr» sein, damit die gesamte Aussage «wahr» ist. Sobald eine Teilaussage «falsch» ist, ist die ganze Aussage falsch.
Die Ergebnisse von Vergleichen in der Mathematik ergeben ebenso die Aussage «wahr oder «falsch». Aus diesem Grund können Gleichungen und Ungleichungen als Parameter einer UND -Verknüpfungen dienen. (Abbildung 48).

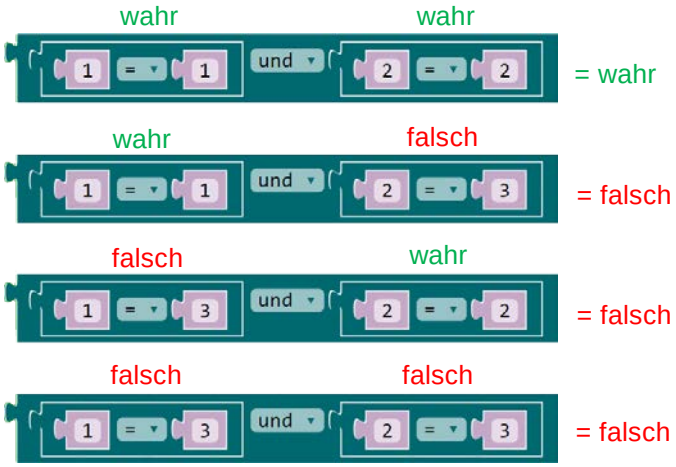


Abbildung 48 – Bei einer UND-Verknüpfung müssen alle Teilaussagen «wahr» sein, damit die gesamte Aussage «wahr» ist.

ODER (OR):
Bei einer ODER-Verknüpfung muss mindestens eine Teilaussage «wahr» sein, damit die gesamte Aussage «wahr» ist (Abbildung 49).

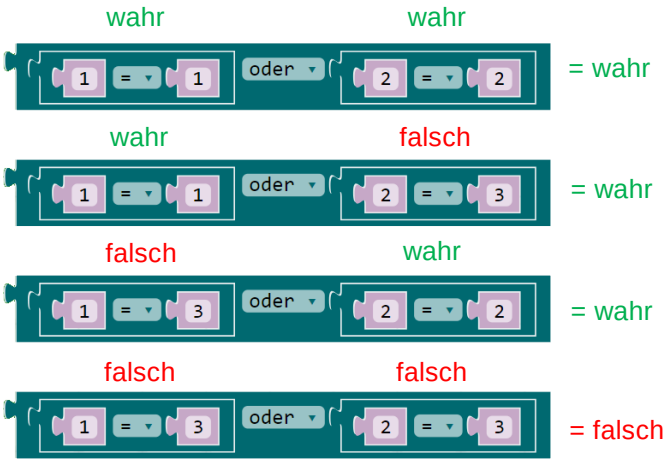


Abbildung 49 – Bei einer ODER-Verknüpfung muss mindestens eine Teilaussage «wahr» sein, damit die gesamte Aussage «wahr» ist.

NICHT (NOT):
Bei einem NICHT wird die Aussage immer ins Gegenteil gesetzt. Dabei können beliebig viele NICHT-Blöcke kombiniert werden. Zwei NICHT heben sich dabei wieder auf (Abbildung 51). Aussagen der Booleschen Algebra können beliebig miteinander kombiniert und verschachtelt werden. Dabei wird die Aussage von innen nach aussen berechnet (Abbildung 50). Es gibt noch weitere Verknüpfungen in der Booleschen Algebra, wie beispielsweise NICHT-UND (NAND), NICHT-ODER (NOR), EXKLUSIVES-ODER (XOR) usw. Mit den beiden Grundbausteinen NICHT sowie UND kann jedoch jede andere Aussage der Booleschen Algebra konstruiert werden.

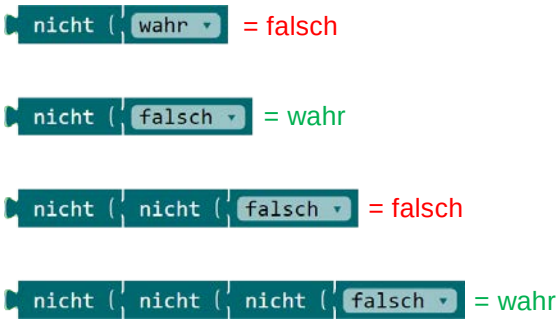


Abbildung 51 – Durch NICHT wird die Aussage ins Gegenteil gesetzt. Zwei NICHT-Blöcke heben sich wieder auf.

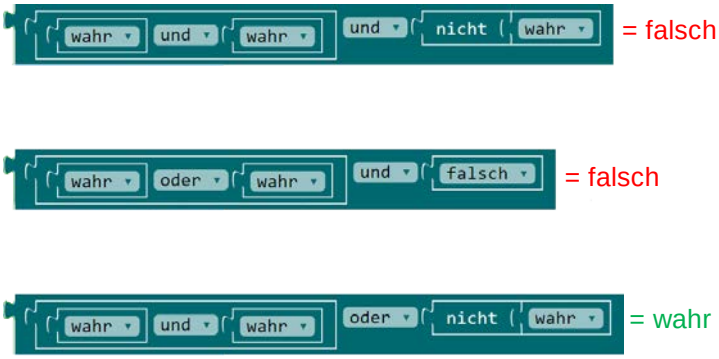


Abbildung 50 – Aussagen der Booleschen Algebra können beliebig miteinander verschachtelt werden.

Die Boolesche Aussagenlogik wird vor allem beim Abfragen in Schleifen und Bedingungen verwendet (Abbildung 52). (a) Die Zeichenfolge wird dann gezeigt, wenn die Taste A oder B oder beide gedrückt sind. (b) Die Zeichenfolge wird dann gezeigt, wenn die Temperatur zwischen 18° und 24° liegt. (c) Das Symbol wird nur dann gezeigt, wenn die Taste A gedrückt ist und die Taste B nicht.

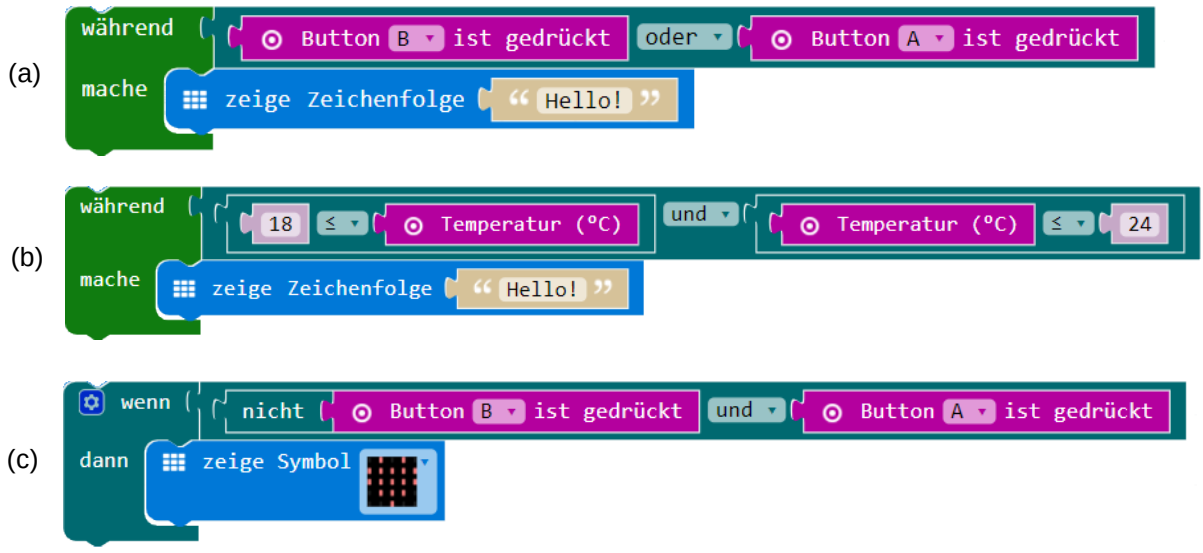


Abbildung 52 – Ausdrücke der Booleschen Algebra werden vor allem bei Bedingungen und Schleifen verwendet.

2.4.10 Unterprogramm/Funktion

Um eine bestimmte Sequenz mehrfach benutzen bzw. wiederverwenden zu können und um das gesamte Programm kompakter zu gestalten, werden Unterprogramme verwendet. Dabei werden Sequenzen zu einer Anweisung zusammengefasst. In der Blockprogrammierung haben Unterprogramme zwei verschiedene Blöcke (Abbildung 53): den Definitionsblock (links) und den dazugehörigen Anweisungsblock (rechts). Der Definitionsblock steht für sich alleine im Programmcode. Innerhalb des Definitionsblocks sind alle Anweisungen des Unterprogramms enthalten. Über den Anweisungsblock kann das Unterprogramm aufgerufen und die darin enthaltenen Anweisungen ausgeführt werden.

Abbildung 53 – Der Definitionsblock und dazugehörige Anweisungsblock eines Unterprogramms.

Unterprogramme werden auch als «Methoden» oder «Funktionen» bezeichnet. Im Grunde führt jede Anweisung ein Unterprogramm aus. Existierende Anweisungen in einer Programmierumgebung sind bereits als Unterprogramme vorprogrammiert und stehen als «Library» zur Verfügung. Die Anweisung «spiele Melodie Dadadum» in Abbildung 24 besteht aus mehreren einzelnen Anweisungen: jede einzelne Note des Lieds. In der Programmierumgebung ist aber nur noch der Anweisungsblock sichtbar und nicht der Definitionsblock, da dieser Teil der Library ist.
In Abbildung 54 wurde ein eigenes Lied «meinLied» als Unterprogramm implementiert. Das Lied kann nun an zwei verschiedenen Orten im Programm über den dazugehörigen Anweisungsblock «Funktion aufrufen meinLied» abgespielt werden: innerhalb «beim Start», sowie «wenn Knopf A gedrückt». Ohne ein Unterprogramm müsste dasselbe Lied an beiden Orten kopiert werden.

Durch Abstraktion identifiziert man eine Sequenz, die als Unterprogramm implementiert werden kann. Dabei muss man sich die Frage stellen, welche Anweisungen essenziell sind für das Unterprogramm.

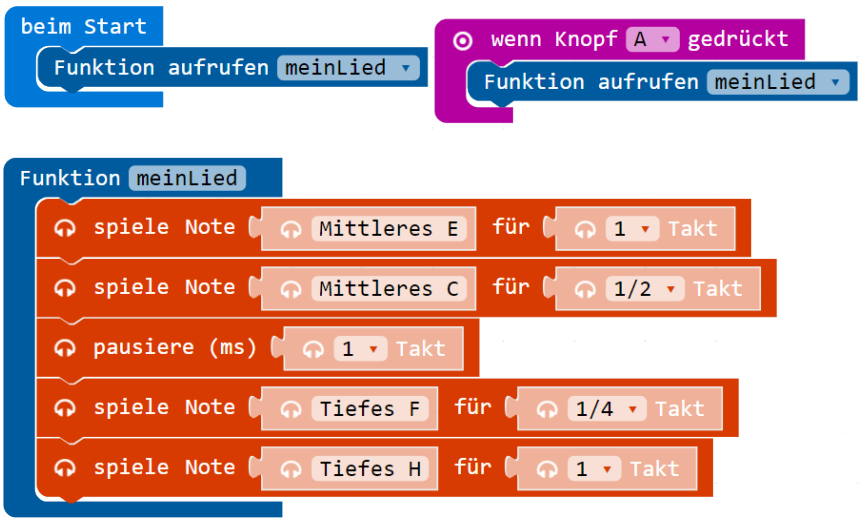


Abbildung 54 – Ein eigenes Lied als Unterprogramm implementiert. Über die Anweisungsblöcke kann das Lied von verschiedenen Stellen im Programmcode aufgerufen werden.

2.5 Implementierung

Eine wichtige Kompetenz in der Informatik ist das strukturierte Vorgehen bei der Umsetzung einer Projektidee. Dabei wird ein Problemlöseprozess durchschritten, der aus einer Reihe von iterativen Schritten besteht, um ein funktionierendes Produkt zu entwickeln. Im projektorientiertem Unterricht steht der gesamte Problemlöseprozess im Zentrum und nicht nur die technische Umsetzung wie z.B. die Programmierung. Der Problemlöseprozess wird im Didaktikteil beschrieben (Kapitel 3.1). Dieses Kapitel widmet sich einem Teil des Problemlöseprozesses, der Implementierung, bzw. der technischen Umsetzung. Beim einer «top-down»-Vorgehensweise möchte man eine bereits bestehende Projektidee konkret umsetzen. Im folgenden Beispiel wird das gemacht.

Projektidee: Eine Heiss-Kalt-Wasserwaage
Problem: Wenn man ein Bild möglichst waagrecht mit zwei Nägeln aufhängen möchte, ist es schwierig, gleichzeitig das Bild zu halten, im richtigen Winkel auf die die Wasserwaage zu schauen und die Wand für die Nägel zu markieren.
Lösung: Eine Wasserwaage, die akustisch und über eine Lampe anzeigt, wenn das Bild gerade ist. Dabei soll das Feedback ähnlich wie bei einem «heiss-kalt»-Spiel sein: Die Lampe soll Grün leuchten und der Ton stumm sein, wenn die Waage gerade ist. Je schräger, desto röter die Lampe und desto höher der Ton.

Implementierung am Beispiel micro:bit:
Um die Komplexität zu reduzieren, muss dabei die Projektidee in Teilprobleme zerlegt werden (Dekomposition).

Teilproblem I: Der micro:bit soll detektieren, wenn sein Rollwinkel geneigt ist. Die Lage des micro:bits ist eine Information von aussen, also ein Sensor. Tabelle 3 beschreibt die wichtigsten Sensoren. Der Lagesensor sieht nach dem geeigneten Sensor aus. Am besten wird dieser einfach einmal getestet und geschaut, ob er sich eignet und was für Werte er liefert. Dabei wird ein Testcode implementiert. Für den micro:bit gibt es zwei spezielle Ereignisse «nach links zeigen» und «nach rechts zeigen». Das sieht vielversprechend aus und wird getestet (Abbildung 55):

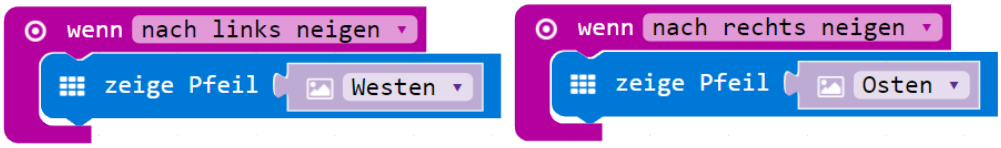
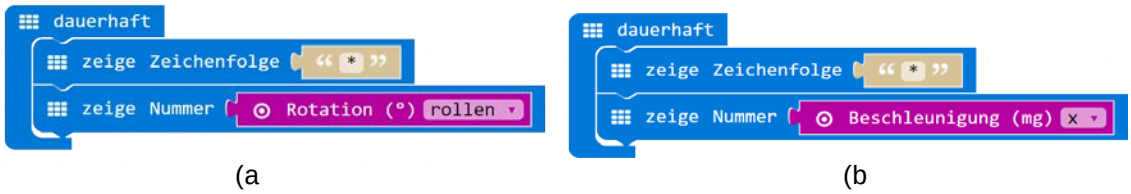


Abbildung 55 – Testen des Lagesensors mit Ereignissen.

Beim Testen wird festgestellt, dass das Ereignis erst dann ausgelöst wird, wenn der micro:bit bei einem Winkel von 45° nach links oder rechts gekippt wird. Für die Wasserwaage muss aber bereits eine geringe Abweichung von der Horizontalen detektiert werden. Also eignet sich die Implementierung über Ereignisse nicht. Ein zweiter Versuch ist über das Polling, d.h. die Werte werden in einer Endlosschleife immer abgefragt. Der Lagesensor besteht aus drei Sensoren: Gyroskop, Beschleunigungssensor und Kompass. Das Gyroskop und der Beschleunigungssensor werden oftmals komplementär verwendet, da sie beide einen Winkel messen können, aber über unterschiedliche physikalische Grössen (Drehgeschwindigkeit vs. Beschleunigung). Es werden beide Sensoren getestet: Abbildung 56(a) Der Rollwinkel über ein Gyroskop und (b) über einen Beschleunigungssensor. Beim Testen sieht man, dass beide Sensoren den Rollwinkel angeben und die horizontale Ausrichtung detektieren. Es scheint jedoch einfacher zu sein, über das Gyro-

skop die Horizontale zu finden als beim Beschleunigungssensor, der feinere Schwankungen in der Messung aufweist und so der Nullpunkt oftmals nicht genau entdeckt wird. Das Gyroskop soll es somit sein.



(a (b

Abbildung 56 - Testen des Rollwinkels. (a) Über ein Gyroskop. (b) Über ein Beschleunigungssensor.

Teilproblem II: Der micro:bit soll ein akustisches Feedback zum Rollwinkel geben. Der micro:bit muss das akustische Signal ausgeben, d.h. es wird ein Aktor benötigt. Tabelle 4 beschreibt die wichtigsten Aktoren. Der Piezo-Buzzer scheint sich dafür zu eignen. Er benötigt einen digitalen Output. In der micro:bit-Challenge Nr. 8 erfährt man, dass ein Buzzer nur über Pin P0 angeschlossen werden kann (Abbildung 57(a)). Über den Testcode in Abbildung 57(b) kann man den Wertebereich des Buzzers testen, d.h. die verschiedenen Frequenzen, welche die Töne erzeugen. Es wird entschieden, über zwei Oktaven zu gehen, vom Tiefen C (131 Hz) bis zum Hohen C (523 Hz).

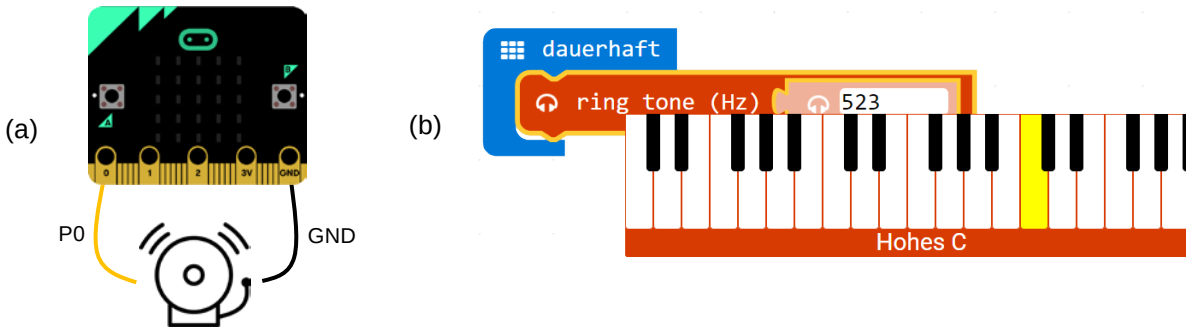


Abbildung 57 – (a) Das Anschliessen eines Buzzers. (b) Testen der verschiedenen Töne.

Um ein akustisches Feedback zum Messwert des Gyroskops zu geben, muss der Wertebereich vom Gyroskop auf denjenigen des Buzzers «gemappt» werden. Das Gyroskop hat ein Wertebereich von -180° (eine Seite) bis +180° (andere Seite). Der Ton soll jedoch für beide Seiten symmetrisch ausgegeben werden, somit kann der absolute Messwert verarbeitet werden. Abbildung 58 zeigt den Programmcode für das Mapping des Gyroskops auf den Buzzer. Eine Variable «Rollwinkel» speichert den absoluten Messwert des Gyroskops, was den Wertebereich von 0° bis 180° ausmacht. Im Falle, dass die Horizontale nicht gefunden wurde («wenn 0 < Rollwinkel»), wird der Rollwinkel auf die Frequenz des Buzzers «gemappt» (siehe auch Kapitel 2.3.4). Wenn die Horizontale gefunden wurde, wird der Buzzer über die Anweisung «schreibe digitalen Wert von Pin P0 auf 1» ausgeschaltet.

Teilproblem III: Der micro:bit soll ein visuelles Feedback zum Messwert geben. Dabei soll ein Licht graduell von Grün auf Rot wechseln. Der micro:bit steuert ein Licht, somit ist es ein Aktor. Tabelle 4 beschreibt die wichtigsten Aktoren. Eine RGB-LED kann mehrere Farben ausgeben und benötigt drei analoge Output-Pins für jede Farbe. Der Buzzer ist jedoch bereits auf Pin P0 angehängt, da stehen nur noch zwei Pins zur Verfügung. Das macht aber nichts, da das Farbspektrum von Rot zu Grün wechseln soll. Die blaue Farbe wird somit nicht benötigt und die RG-LED wird wie in Abbildung 59(a) angeschlossen. Abbildung 59(b) zeigt den gesamten Programmcode für die Heiss-Kalt-Wasserwaage. Der Gradient für die RG-LED wird ebenso über die Mapping-Funktion implementiert. Dabei wurde der Wertebereich des Gyroskops etwas

verkürzt (1°-90°), damit es schneller von Grün zu Rot wechselt. Ein kleiner Winkel hat einen hohen Grünanteil (P1 auf 1023) und ein grosser Winkel einen hohen Rotanteil (P2 auf 1023) im Licht.

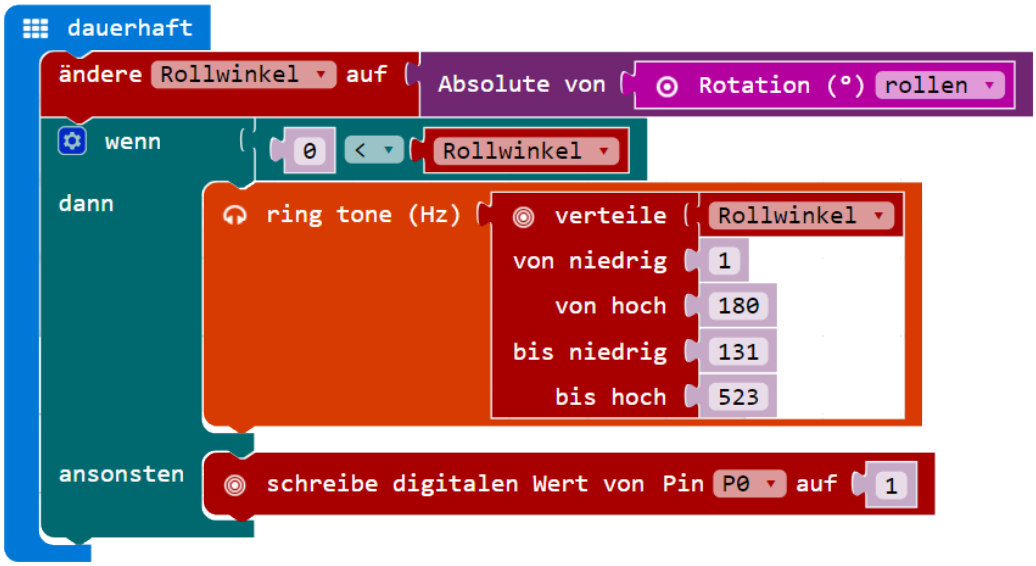


Abbildung 58 – Das Mapping des Gyroskops auf den Buzzer.

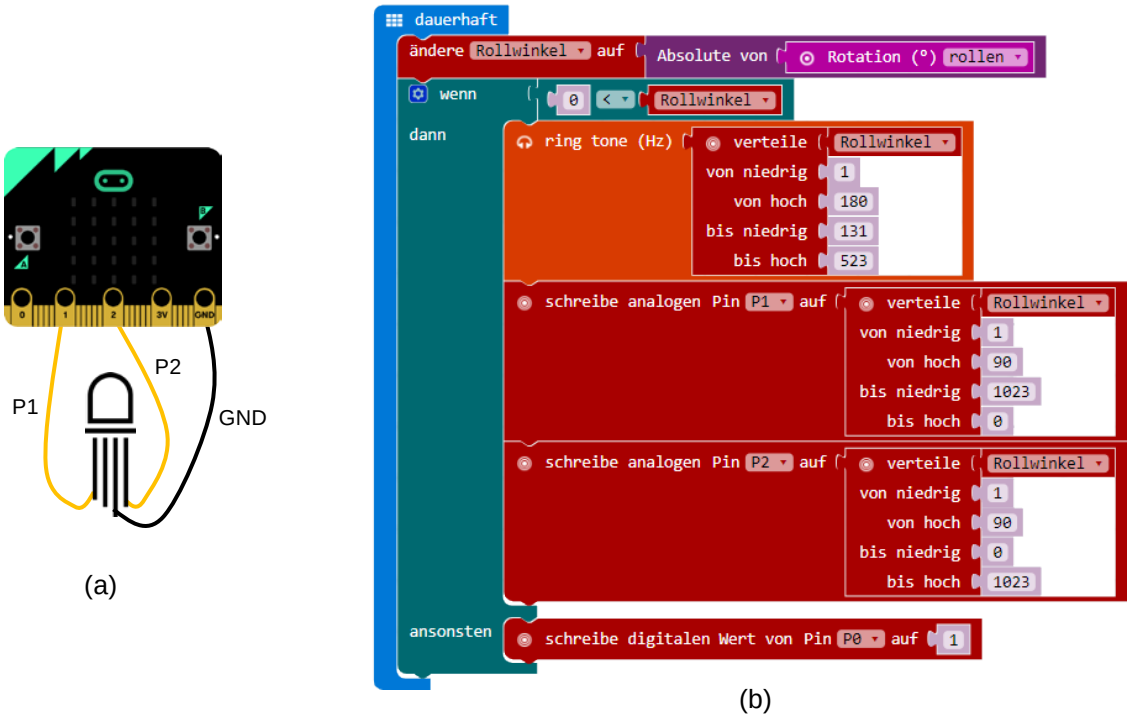


Abbildung 59 - (a) Anschluss einer RG-LED. (b) Der gesamte Programmcode für die Heiss-Kalt-Wasserwaage.

Dieses Projekt wurde also in drei Teilprobleme aufgeteilt und schliesslich wurde alles wieder zusammengesetzt. Es ist von Vorteil, die einzelnen Teilprobleme so zu implementieren und zu testen, dass sie einzeln funktionieren. Das hilft auch bei der Fehlersuche.

2.6 Fehlersuche (Debugging)

Nur aus Fehlern lernt man! Eine wichtige Kompetenz ist die Fehlersuche, oder auch Debugging genannt. Im Gegensatz zum Programmieren in rein virtuellen Welten (z.B. Scratch), kommt im Physical Computing noch die physische Welt dazu. Wenn etwas nicht so funktioniert, wie es sollte, kann sich der Fehler dabei in der Hardware (Elektronik) und/oder in der Software verstecken. Die häufigsten Fehler lassen sich jedoch schnell finden, wie folgend am Beispiel von micro:bit und Calliope gezeigt wird:

- **Der Strom fehlt.** Hört sich zwar trivial an, ist aber häufig das Problem. Das USB-Kabel oder das Batteriefach ist nicht angeschlossen oder die Batterien sind leer.
- **Der falsche Programmcode wurde hochgeladen.** Nach dem «Herunterladen» im Browser wird das aktuelle Programm mit dem Dateinamen des Projektes gespeichert. Wenn bereits eine Datei unter dem gleichen Namen vorhanden ist, wird eine Zahl am Ende des Dateinamens inkrementiert. Wenn der Datei-Explorer nicht aktualisiert ist, kann es sein, dass man versehentlich einen alten Programmcode auf den micro:bit oder Calliope hochlädt.
- **Der falsche Pin wurde gewählt.** In de, Anweisungen, z.B. «schreibe digitalen Wert von Pin P0» oder «lese analoge Werte von Pin P1» sind die falschen Pins gewählt. Dies geschieht sehr häufig, wenn man vergisst, den Standardwert in der Anweisung zu ändern.
- **Die elektronische Schaltung ist nicht korrekt.** Die Krokodilklemme hat sich gelöst, wurde verschoben oder die Verbindung fehlt vollständig. Jeder Sensor oder Aktor muss einen geschlossenen Schaltkreis haben, d.h. mindestens zwei Kabel müssen vorhanden sein. Natürlich müssen sie auch korrekt verbunden sein. Bei den meisten Komponenten muss die Polarität beachtet werden!
- **Der Programmcode wird gar nie ausgeführt.** Wurde eine Bedingung implementiert, die gar nie eintrifft? Beispielsweise kann die Bedingung «wenn Lichtstärke = 34» unter Umständen gar nie eintreffen. Sensoren weisen immer Schwankungen in der Messung auf. Es sollte demzufolge eher ein Messbereich gewählt werden z.B. «wenn 30 < Lichtstärke UND 40 > Lichtstärke».
- **Der falsche Wertebereich wurde gewählt.** Geben die Sensoren wirklich diese Messwerte aus, die im Programmcode angenommen wurden? Wertebereiche der Sensoren muss man prüfen.

Eine systematische Vorgehensweise ist bei der Fehlersuche von Vorteil. Wann hat das System zum letzten Mal richtig funktioniert? Was wurde seitdem verändert? Die einzelnen Komponenten im System sollen dabei separat getestet werden. Das System soll immer inkrementell nach dem Testen mit neuen Komponenten und Funktionen erweitert werden. So kann man Änderungen wieder rückgängig machen und schauen, wo das Problem liegt. Viele Programmierumgebungen stellen Tools fürs Debuggen zur Verfügung. Diese beinhalten z.B. einen Emulator des Systems und das Hervorheben (Highlighten) des aktuell ausgeführten Programmcodes. Der Emulator ist ein sehr gutes Tool, um Code schnell zu testen, ohne diesen jedes Mal auf die Plattform hochladen zu müssen.
In klassischen Programmierumgebungen gibt es den Output auf der Konsole. Das ist die Möglichkeit, Mitteilungen in Form eines Logs auszugeben. Beim micro:bit oder Calliope gibt es leider keine solche Konsole, ausser vielleicht das LED-Display, was aber die Ausführung des Codes verlangsamt. Zum Debuggen benutzt man dann oftmals LEDs, die in gewissen Fällen leuchten oder nicht.
Das Wichtigste ist aber auch, Programmcode immer zu kommentieren. Jede Programmierumgebung bietet eine solche Funktionalität an. Oftmals weiss man bereits nach einigen Tagen nicht mehr, was man genau gemacht hat und muss sich wieder eindenken. Das geht auch professionellen Programmiererinnen und Programmierern so! Und schliesslich ist es auch immer gut möglich, dass das eine oder andere elektronische Bauteil beim Experimentieren kaputtgegangen ist.

3 Zum Unterricht

Seymour Papert, Mathematiker, Pädagoge und Mitentwickler der Programmiersprache Logo, war ein Vertreter der konstruktivistischen Lerntheorie. Diese besagt, dass Lernende ihr Wissen durch das Abgleichen neuer Erfahrungen mit dem eigenen Vorwissen selber konstruieren. Der soziale Kontext spielt dabei eine grosse Rolle: Die Kollaboration mit anderen fördert den Lernprozess. Bereits seit den 1970er-Jahren propagierte Papert die umfassende Integration von Computertechnologie in den Unterricht. In einer Publikation aus dem Jahr 1971 mit dem Titel «Twenty Things to Do with a Computer» beschreibt er verschiedene Projektideen für den Unterricht, u.a. das Komponieren von Musik, die Steuerung von Marionetten, die Kreation von Filmen, Modellierung usw. Den Schulcomputer stellte er sich dabei wie folgt vor:

> *The school computer should have a large number of output ports to allow the computer to switch lights on and off, start tape recorders, actuate slide projectors and start and stop all manner of little machines. There should also be input ports to allow signals to be sent to the computer. [..]*
> *In our image of a school computation laboratory, an important role is played by numerous "controller ports" which allow any student to plug any device into the computer. […] The laboratory will have a supply of motors, solenoids, relays, sense devices of various kinds, etc. Using them, the students will be able to invent and build an endless variety of cybernetic systems.* (S. 19-20, S.L. Martinez & G. Stager, «Invent to Learn», Constructing Modern Knowledge Press, 2013.)

Es ist faszinierend, dass Papert bereits vor über 40 Jahren den Schulcomputer als ein Physical-Computing-Tool betrachtet hat. Der Computer sollte für die Schülerinnen und Schüler einfach ein weiteres «Material» sein, wie Papier und Massstab. «Making» befasst sich mit dem Design und dem Kreieren von Dingen mit traditionellen und neuen Materialien und Werkzeugen. Physical-Computing-Tools sind neben der digitalen Fabrikation (3D-Drucker, Lasercutter, CNC-Fräsen) ein wichtiger Bestandteil dieser neuen, modernen Werkzeuge. In den letzten Jahren hat Making auch Einzug in den Schulunterricht gehalten. Die Didaktik für Making in der Schule ist sich am Entwickeln und es werden noch viele Jahre Forschung und Erfahrung in diesem Bereich benötigt. Die Natur von Making bestimmt aber, dass die Arbeitsweise kollaborativ, projektorientiert, «open-ended», kreativ und iterativ ist. Viele dieser didaktischen Prinzipien decken sich mit denjenigen von Papert. Deshalb wird er oftmals auch als Vater des Maker-Movement bezeichnet.

3.1 Problemlöseprozess

Egal, ob im textilen, bildnerischen- oder technischen Gestalten, in der Musik usw.; Überall dort, wo etwas Neues geschaffen wird, wird ein Problemlöseprozess durchlaufen. Jedes Fachgebiet hat seine eigenen Modelle und Methodiken, doch im Grunde beschreiben sie einen iterativen Prozess, der von der Idee zum Produkt führt. Die Modelle weisen unterschiedliche Bezeichnungen und Granularitäten für die einzelnen Phasen auf. Der Einfachheit halber werden hier drei Hauptphasen beschrieben: Think, Make und Improve (Abbildung 60) (S. 52-54, S.L. Martinez & G. Stager, «Invent to Learn», Constructing Modern Knowledge Press, 2013).

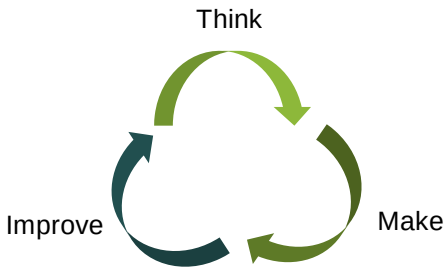


Abbildung 60 - Ein iterativer Problemlöseprozess.

In diesen drei Phasen können folgende Tätigkeiten stattfinden (übersetzt aus dem Englischen von D.A., S. 52-54, S.L. Martinez & G. Stager, «Invent to Learn», Constructing Modern Knowledge Press, 2013):

- **Think:** Brainstorming, Diskutieren, Prognostizieren, Sammeln von Material, Expertise identifizieren, Gruppen bilden (oder alleine arbeiten), Zielsetzung, Skizzieren, Abgrenzen, Diagramme zeichnen, Recherchieren, Planen.
- **Make:** Spielen/Ausprobieren, Bauen, Basteln, Kreieren, Programmieren, Experimentieren, Konstruieren, Auseinandernehmen, Vorgehensweisen und Materialien testen, andere beobachten, Code kopieren, Code teilen, Prozess dokumentieren, Probleme suchen und finden, Fragen stellen, Fehler beheben.
- **Improve:** *Wenn es nicht funktioniert oder das Team steckenbleibt:* Recherchieren, Ausdiskutieren im Team, Diskutieren mit Kollegen, das Problem von einem anderen Blickwinkel betrachten, andere Materialien verwenden, einzelne Komponenten des Projektes verändern, diskutieren, ob und wie ein ähnliches Problem zu einem früheren Zeitpunkt gelöst wurde, herumspielen, ein ähnliches Projekt finden zum Analysieren oder Auseinandernehmen, eine Expertin oder einen Experten fragen, cool bleiben, frische Luft schnappen, darüber schlafen.
  Wenn das Team «fertig» ist: Möglichkeiten finden, um das Produkt zu verbessern oder weiterentwickeln. Fragestellungen: Wie kann das Produkt so gestaltet werden, dass es schneller, langsamer, besser, genauer, schöner, ökologischer, cooler, stärker, smarter, flexibler, grösser, kleiner, effizienter, kostengünstiger, verlässlicher, leichter, eleganter, einfacher zu benutzen ist?

Schliesslich soll das Produkt kommuniziert und die Expertise geteilt werden. Gemäss Papert und der konstruktivistischen Lerntheorie fördert der Kreislauf «Making» und «Making things better» das Verständnis.

Ein weiteres GMI21-Themendossier befasst sich mit dem didaktischen Konzept von Making: «Making macht Schule – Informatik begreifbar erleben im Zyklus 2» (Pädagogische Hochschule St. Gallen).

3.2 Projektorientierter Unterricht

Ein Projekt in diesem didaktischen Kontext sollte möglichst «open-ended» und bedeutsam sein und einen persönlichen Bezug zu den Schülerinnen und Schülern haben. Ein Open-ended-Projekt bedeutet, dass es keine Musterlösung zum Problem gibt, d.h. es kann auf viele verschiedene Arten gelöst werden. Die Projektdauer kann sich dabei über eine oder mehrere Lektionen erstecken oder auch über ein Semester.

Was macht ein gutes Projekt aus? (Teilweise übernommen aus dem Englischen und ergänzt von D.A., S. 58-59, S.L. Martinez & G. Stager, «Invent to Learn», Constructing Modern Knowledge Press, 2013)

- **Zweck und Relevanz:** Hat das Projekt einen persönlichen Bezug und Relevanz? Weckt das Projekt genügend Interesse, um Zeit, Energie und Kreativität darin zu investieren?
- **Komplexität:** Ein gutes Projekt beinhaltet verschiedene Themen und knüpft am Vorwissen an.
- **Neuheit:** Es gibt selten ein ultimatives Projekt, das so bahnbrechend ist, dass man es mit allen Schülerinnen und Schülern jedes Jahr wiederholen muss. Am besten ist es, einander die Erfahrungen aus verschiedenen Projekten zu vermitteln und das Wissen miteinander zu teilen.
- **Wiederverwendbarkeit:** Schülerinnen und Schüler sollen etwas erschaffen, das sie untereinander teilen und wiederverwenden können (z.B. Programmcode, Konstruktionen usw.) Dies fördert auch die Motivation und Relevanz der Arbeit jedes einzelnen.

- **Gendergerecht:** Für einen gendergerechten Unterricht ist es ratsam, Themen zu wählen, die verschiedene Interessen der Schülerinnen und Schüler ansprechen. Projekte im Physical Computing bieten dafür gute Möglichkeiten, da sie an den Schnittstellen zu anderen Disziplinen angesiedelt sind. So kann man Projekte zum Thema Musik, Naturwissenschaften, Mathematik und Sprache umsetzen oder interaktive Kunstinstallationen bauen und E-Textilien entwerfen. Wenn die Projekte open-ended sind, viele verschiedene Materialien beinhalten und an die Lebenswelt der Schülerinnen und Schüler anknüpfen, holt man sicher ein breites Zielpublikum ab.

Was sind die Rahmenbedingungen?

- **Zeit:** Genügend Zeit muss bereitgestellt werden, damit der Problemlöseprozess iterativ durchlaufen werden kann.
- **Fehlerkultur:** Fehler sind ein wichtiger Bestandteil des Lernprozesses. Die Schülerinnen und Schüler sollen dazu motiviert werden, Risiken einzugehen und einfach etwas auszuprobieren. Es ist auch ganz normal, dass etwas dabei kaputtgehen kann. Fehler sollen keinen negativen Faktor in der Beurteilung des Projektes ausmachen, im Gegenteil.
- **Kollaboration:** Programmieren lernt man am besten, wenn man bestehenden Programmcode analysiert, abändert und ergänzt. Code von anderen zu kopieren, um Ideen nachzubauen, ist eine wichtige Arbeitsweise in der Informatik und im Making.
- **Zugang zu Materialien:** Es ist sehr wichtig, dass die Schülerinnen und Schüler möglichst uneingeschränkten Zugang zu den Materialien (Software, Hardware, Bastelmaterial usw.) haben und dass die Infrastruktur funktioniert (WLAN, Computer usw.), am besten auch ausserhalb der offiziellen Lektionen.

Um sich in ein neues Thema einzuarbeiten, ist es hilfreich, zunächst Schritt-für-Schritt-Tutorials durchzuarbeiten. Für jede Physical-Computing-Plattform findet man genügend solcher «Getting started»-Anleitungen. Es ist jedoch wichtig, sich mit der Zeit von diesen Anleitungen zu lösen. Durch blindes Nachbauen und Abschreiben des Programmcodes kommt man nicht zu dem Konzeptwissen, das benötigt wird, um eigenständige Lösungen zu implementieren. Die Aktivitäten sollten nach den anfänglich stark geleiteten Tutorials schnell open-ended werden, d.h. es sollte möglich sein, eigene Lösungen umzusetzen.

Bei der Vermittlung von Kompetenzen sind die Lehrpersonen mit einer riesigen Schere von Vorwissen konfrontiert. Die Schritt-für-Schritt-Tutorials eignen sich hierfür sehr gut. Diejenigen Schülerinnen und Schüler, die mehr Unterstützung benötigen, können länger geleitete Tutorials bearbeiten. Es gilt dabei zu differenzieren und herauszufinden, wer konkrete Schritt-für-Schritt-Anleitungen braucht und wer durch freies Ausprobieren zum Ziel kommt.
Die micro:bit- und Calliope-Challenge-Cards (Kapitel 4.1 und 4.2) sind so aufgebaut, dass sie anfänglich als Schritt-für-Schritt-Tutorials bearbeitet werden können. Jede Challenge behandelt unterschiedliche Konzepte, wie z.B. digitaler Output, analoger Input, Polling, Ereignisse usw., sowie verschiedene Sensoren und Aktoren. Diejenigen, die nach den ersten Karten «das Prinzip begriffen haben», können eigene Projekte definieren und die Karten zum Nachschlagen nutzen. Andere können alle Challenges komplett durcharbeiten, um die wichtigsten Konzepte einmal gesehen zu haben.

Projektorientierter Informatikunterricht widerspiegelt die Realität sowie die künftigen Arbeitsprozesse im Berufsleben. Die Frage stellt sich, wie man Projekte der Schülerinnen und Schüler bzw. die erworbenen Kompetenzen beurteilt.

> *[…] Kompetenz zeigt sich also erst dann, wenn wir das erworbene Wissen und Können in neuen Zusammenhängen anwenden und vergleichbare Aufgaben lösen können. Dementsprechend ist auch der Lehr- und Lernprozess erst dann abgeschlossen, wenn Schülerinnen und Schüler neu erworbene Kompetenzen in Form von Handlungen zeigen können (dies auch in Form von Sprech- und Schreibhandlungen oder medialen Produkten). Wenn sich der kompetenzorientierte Unterricht auf das Können ausrichtet, soll sich dementsprechend auch die Beurteilung daran orientieren, wie gut die Schülerinnen und Schüler ihr neu erworbenes Wissen und Können in neuen Situationen anwenden können. Zum kompetenzorientierten Lernen gehört immer auch eine kompetenzorientierte Beurteilung. […] Während formative Beurteilungsformen dazu beitragen sollen, den Lernprozess zu optimieren und aus Fehlern zu lernen, überprüft die summative Beurteilung im Sinne einer abschliessenden Bewertung den Erfüllungsgrad der definierten Lernziele. Während des Lernprozesses steht zunächst die formative Beurteilung im Vordergrund, erst gegen Schluss einer Lernphase folgt die summative Beurteilung.* (S. 30-34, I. Schrackmann et al., Wegleitung «Medien und Informatik» für Lehrpersonen der Primarstufe (5./6. Klasse), Amt für Volksschulen und Sport des Kantons Schwyz, 2018, http://link.phsz.ch/mi-56).

Nicht nur für die Beurteilung, sondern auch für die eigenen Notizen ist es ratsam, das Projekt fortlaufend in Form eines Portfolios zu dokumentieren. Dabei sollen auch Erkenntnisse aus Fehlern aufgeführt werden. Programmcode soll in der Programmierumgebung kommentiert und erklärt werden können.

3.3 Die Kompetenzen der Lehrperson

Durch den Einzug der Informatik als neue Disziplin in der Volksschule stellen sich neue Chancen und Herausforderungen. Man muss sich bewusst sein, dass der Informatikunterricht auf dieser Schulstufe für alle Beteiligten (Pädagogische Hochschulen, Lehrpersonen, Bildungspolitiker, Eltern, Schülerinnen und Schüler, Schulleitungen, Didaktiker, Wissenschaftler, IT-Dienste usw.) neu ist. Unsere Nachbarländer durchlaufen zurzeit ähnliche Prozesse und die Dynamik auf dem Gebiet ist gut spürbar. Projekte wie micro:bit und Calliope, die eigens für die Volksschule entwickelt worden sind, zeugen von dem Impuls, welche die Lehrplanreformen bewirken.

Projektorientierter Unterricht im Physical Computing ist unvorhersehbar. Die Schülerinnen und Schüler müssen Probleme lösen, für die es keine Musterlösung gibt. Lehrpersonen finden sich dabei eher in der Rolle eines Coaches wieder. Durch systematisches Vorgehen, Erfahrung und die richtigen Fragen kann man dem Problem auf die Spur kommen und gemeinsam eine Lösung finden. Dieser Weg ist ein wichtiger Teil der Arbeit, nicht bloss das funktionierende Endprodukt. Dieses Heft versucht, das für die Lehrperson benötigte Konzeptwissen zu vermitteln, um die richtigen Fragen zu stellen. Natürlich muss man sich auch immer wieder selber weiterbilden. Es gibt sehr viel gute Literatur zu diesem Thema (Tabelle 8).

Die Lehrperson muss sich bewusst sein, dass sich die Technologie ständig ändert, die Infrastruktur heterogen sein kann (insbesondere bei «Bring Your Own Device»-Lösungen) und sie nicht alles wissen kann. Es ist gut möglich, dass einige Schülerinnen und Schüler die Nase vorn haben. Ihre Expertise ist wertvoll und soll für die Hilfestellung der Mitschülerinnen und Mitschüler miteinbezogen werden. Das entlastet auch die Lehrperson. Die reale Welt ist «messy», und wenn man die Schülerinnen und Schüler darauf vorbereiten möchte, muss man das Unvorhersehbare zulassen. Und das funktioniert kaum über standardisierte Inhalte und Aktivitäten. All dies ist für Lehrpersonen eine grosse Herausforderung. Als kleiner Trost sei hier gesagt: Es sitzen alle im gleichen Boot und niemand weiss wirklich, wie der Hase läuft!

3.4 Die Toolkits

Die technischen Tools spielen im Informatikunterricht eine wichtige Rolle. Bereits in den 1970er-Jahren hat Seymour Papert propagiert, den Computer als Material neben vielen Materialien im Schulzimmer zu betrachten. Die Materialien sind austauschbar und so sind es auch die didaktischen Toolkits.

Wenn es um das Programmieren in der Volksschule geht, wird oft – auch von der Industrie beeinflusst – der Fokus auf gewisse Programmiersprachen gesetzt. Dabei wird die textuelle Programmierung als «richtiges Programmieren» bezeichnet und von der Blockprogrammierung abgesetzt. Diese Unterscheidung macht wenig Sinn, da alles am Schluss so übersetzt wird, dass der Computer das Programm ausführen kann. Dem Computer ist es gleichgültig, ob die Anweisungen grafisch oder textuell eingegeben wurden. Das Wichtigste ist es, dass die Programmierkonzepte verstanden wurden und angewandt werden können, dann hat man die Programmierkompetenzen erworben. Die Programmiersprache soll in erster Linie aus didaktischen Überlegungen gewählt werden und nicht im Hinblick darauf, was zurzeit in der Berufswelt benutzt wird.

Mit dem Aufkommen der Maker-Community und der Einführung des Programmierunterrichts in der Schule wurden in den letzten Jahren viele neue didaktische Tools fürs Physical Computing entwickelt. Mittlerweile ist für jede Anwendung und jedes Budget etwas dabei. Didaktische Tools sind austauschbar, die Konzepte, die sich vermitteln sind über die Tools hinweg allgemeingültig. Bei der Wahl eines Toolkits ist zu empfehlen, dass es idealerweise einen niederschwelligen Zugang bietet, aber in der Komplexität offen ist («low threshold, high ceiling»). Und auch hier gilt: das ultimative Tool, das alle Probleme im Informatikunterricht löst, gibt es nicht!
Tabelle 5 listet eine Auswahl an Toolkits fürs Physical Computing auf. Zu den Toolkits kann (und soll) auch klassisches Material verwendet werden, wie z.B. Holz, Textil, Farbe, Karton, PET-Flaschen, Styropor, LEGO-Bausteine usw. Für die interaktiven Projekte im Physical Computing eignen sich auch leitfähige Materialien, wie z.B. leitfähiger Faden für E-Textilien, leitfähige Farbe, Kupfertape und leitfähiger Teig (siehe Kochrezept, Kapitel 7). In Tabelle 9 ist noch weiteres Zubehör aufgelistet.

Tabelle 5 – Einige Toolkits für Physical Computing.

Produkt	**Link**	**Kommentar**
micro:bit	https://microbit.org/de/	Für die Schule entwickelt
Calliope	https://calliope.cc/	Für die Schule entwickelt
CodeBug	https://www.codebug.org.uk/	Für die Schule entwickelt Geeignet für E-Textilien
OXOCard	http://www.oxocard.ch/	Für die Schule entwickelt Schweizer Produkt
LittleBits	https://www.littlebits.com/	Unterstufe
Arduino	https://www.arduino.cc/	Für Fortgeschrittene
LilyPad	https://www.arduino.cc/en/Main/ArduinoBoardLilyPad/	Arduino-basiert Geeignet für E-Textilien
CircuitPlayground	https://learn.adafruit.com/introducing-circuit-playground/overview	Arduino-basiert Geeignet für E-Textilien
LEGO WeDo	https://education.lego.com/de-de/product/wedo-2	Unterstufe
LEGO Mindstorms EV3	https://www.lego.com/de-de/mindstorms/about-ev3	Mittel-/Oberstufe Komplettes Konstruktionsset
Raspberry Pi	https://www.raspberrypi.org/	Für Fortgeschrittene Linux-Betriebssystem

4 Zum Ausprobieren

Einfache Tutorials für Einsteigerinnen und Einsteiger in Kartenform mit Challenges und Lösungen zum Ausprobieren. Die Karten im A5-Format können beidseitig ausgedruckt werden. Die Challenge ist jeweils auf der Vorderseite und die Lösung auf der Rückseite. Es wird empfohlen, beim ersten Mal die Cards der Reihe nach durchzuarbeiten (Grundlagen danach Einsteiger 1-17). Anschliessend können die Cards auch zum Nachschlagen verwendet werden. Tabelle 6 (BBC micro:bit) und Tabelle 7 (Calliope mini) zeigen eine Übersicht über die Challenges und deren wichtigsten Konzepte mit Links zur Theorie in diesem Heft.

4.1 micro:bit Challenge-Cards

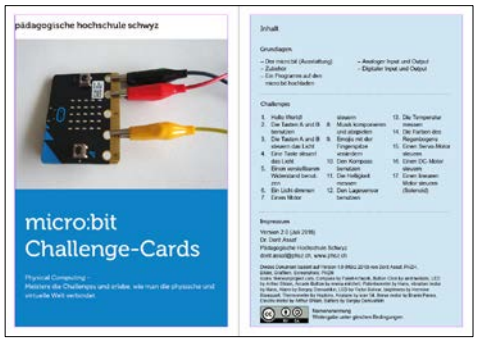


Tabelle 6 – Eine Übersicht über die micro:bit Challenge-Cards

Grundlagenkarte			
Der micro:bit (Ausstattung)		Analoger Input und Output	
Zubehör		Digitaler Input und Output	
Ein Programm auf den micro:bit hochladen			
EinsteigerInnenkarte		**Konzepte**	**Externe Komponenten**
1	Hello World!	Endlosschleife	
2	Die Tasten A und B benutzen	Digitaler Input, Ereignis	
3	Die Tasten A und B steuern das Licht	Digitaler Input Digitaler Output, Ereignis	LED
4	Eine Taste steuert das Licht	Digitaler Input Digitaler Output, Polling	Taste LED
5	Einen verstellbaren Widerstand benutzen	Analoger Input, Polling	Potentiometer
6	Ein Licht dimmen	Analoger Input Analoger Output, Polling	Potentiometer LED
7	Einen Motor steuern	Analoger Input Analoger Output, Polling	Potentiometer Vibrationsmotor
8	Musik komponieren und abspielen	Digitaler Output, Schleife	Buzzer
9	Emojis mit der Fingerspitze verändern	Digitaler Input, Ereignis	
10	Den Kompass benutzen	Analoger Input, Polling	
11	Die Helligkeit messen	Analoger Input, Polling	
12	Den Lagesensor benutzen	Ereignis, Polling	
13	Die Temperatur messen	Analoger Input, Polling	

14	Die Farben des Regenbogens	Analoger Output	RGB-LED
15	Einen Servo-Motor steuern	Analoger Input Mapping	Potentiometer Servo-Motor
16	Einen DC-Motor steuern	Analoger Input Analoger Output	DC-Motor
17	Einen linearen Motor steuern (Solenoid)	Digitaler Output, Ereignis	Linearer Motor

4.2 Calliope mini Challenge-Cards

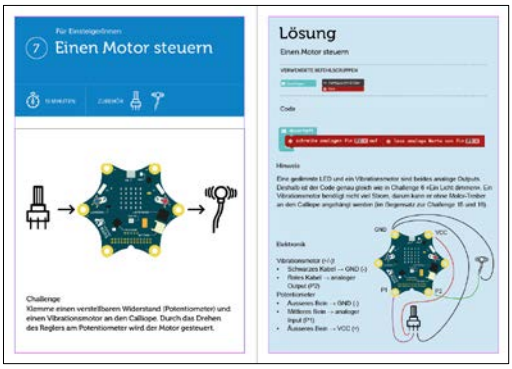


Tabelle 7 – Eine Übersicht der Calliope mini Challenge-Cards

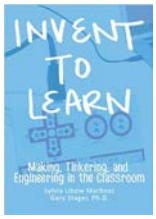
Grundlagenkarte			
Der Calliope mini (Ausstattung)		Analoger Input und Output	
Zubehör		Digitaler Input und Output	
Ein Programm auf den Calliope hochladen			
EinsteigerInnenkarte		**Konzepte**	**Externe Komponenten**
1	Hello World!	Endlosschleife	
2	Die Tasten A und B benutzen	Digitaler Input, Ereignis	
3	Die Tasten A und B steuern das Licht	Digitaler Input, Digitaler Output, Ereignis	LED
4	Eine Taste steuert das Licht	Digitaler Input, Digitaler Output, Polling	Taste LED
5	Einen verstellbaren Widerstand benutzen	Analoger Input, Polling	Potentiometer
6	Ein Licht dimmen	Analoger Input, Analoger Output, Polling	Potentiometer LED
7	Einen Motor steuern	Analoger Input, Analoger Output Polling	Potentiometer Vibrationsmotor
8	Musik komponieren und abspielen	Digitaler Output, Schleife	
9	Farben mit der Fingerspitze verändern	Digitaler Input, Ereignis	
10	Den Kompass benutzen	Analoger Input, Polling	

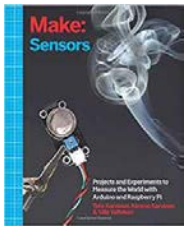
11	Die Helligkeit messen	Analoger Input, Polling	
12	Den Lagesensor benutzen	Ereignis, Polling	
13	Die Temperatur messen	Analoger Input, Polling	
14	Einen Krachometer bauen	Analoger Input, Polling, Bedingung, Boolesche Algebra	
15	Einen Servo-Motor steuern	Digitaler Output	Servo-Motor
16	Einen DC-Motor steuern	Analoger Input, Analoger Output, Mapping	DC-Motor Potentiometer
17	Einen linearen Motor steuern (Solenoid)	Digitaler Output, Ereignis	Linearer Motor

5 Zum Weiterlesen

5.1 Literatur zum Weiterlesen und Nachschlagen

Tabelle 8 – Literatur zum Weiterlesen und Nachschlagen

	**Das Calliope-Buch** Nadine Bergner / Patrick Franken / Julia Kleeberger et al. Taschenbuch: 322 Seiten Verlag: dpunkt, Auflage: 1 (Juli 2017) Sprache: Deutsch ISBN Print: 978-3-86490-468-4	Viele spannende Projektideen und Erklärungen zu Konzepten. Die meisten Projekte können auch mit dem micro:bit umgesetzt werden. 
	**Invent To Learn: Making, Tinkering, and Engineering in the Classroom** Sylvia Libow Martinez/ Gary S. Stage Taschenbuch: 252 Seiten Verlag: Constructing Modern Knowledge Press (Mai 2013) Sprache: Englisch ISBN Print: 978-0989151108	Gutes Buch zur Didaktik von Making in der Schule. 
	**Encyclopedia of Electronic Components Volume 3: Sensors for Location, Presence, Proximity, Orientation, Oscillation, Force, Load, Human Input, Liquid ... Light, Heat, Sound, and Electricity** Charles Platt, Fredrik Jansson Taschenbuch: 256 Seiten Publisher: Maker Media, Inc; Auflage: 1 (5. Mai 2016) Sprache: Englisch ISBN-13: 978-1449334314	Gutes Nachschlagewerk für Physical Computing. Beinhaltet die wichtigsten Sensoren. 

	**Making Things Talk: Die Welt hören, sehen, fühlen** Tom Igoe Taschenbuch: 486 Seiten Verlag: O'Reilly Verlag GmbH & Co. KG; Auflage: 1 (1. April 2012) Sprache: Deutsch ISBN-13: 978-3868991628	Verschiedene spannende Projektbeispiele zu Physical Computing.
	**Make: Elektronik: Eine unterhaltsame Einführung für Maker, Kids und Bastler** Charles Platt Taschenbuch: 472 Seiten Verlag: dpunkt.verlag GmbH; Auflage: 2 (27. Oktober 2016) Sprache: Deutsch ISBN-13: 978-3864903687	Vorwiegend analoge Schaltkreise.
	**Make: More Electronics: Journey Deep Into the World of Logic Chips, Amplifiers, Sensors, and Randomicity** Charles Platt Taschenbuch: 392 Seiten Publisher: Maker Media, Inc; Auflage: 1 (Mai 2014) Sprache: Englisch ISBN-13: 978-1449344047	Analoge und digitale Schaltkreise.
	**Make: Sensors: A Hands-On Primer for Monitoring the Real World with Arduino and Raspberry Pi** Tero Karvinen, Kimmo Karvinen, Ville Valtokari Taschenbuch: 400 Seiten Verlag: Maker Media, Inc; Auflage: 1 (Juni 2014) Sprache: Englisch ISBN-13: 978-1449368104	Für Fortgeschrittene, die Arduino oder Raspberry Pi ausprobieren möchten.

6 Zum Beschaffen

6.1 Empfohlenes Zubehör mit Links zu Online-Shops

Tabelle 9 – Empfohlenes Zubehör mit Links zu Online-Shops.

Produkt	Link Online Shop (Stand März 2018)
BBC micro:bit starter kit 	https://www.kitronik.co.uk/5615-bbc-microbit-starter-kit.html (UK) https://educatec.ch/detail/index/sArticle/1928 (CH) https://shop.pimoroni.com/collections/micro-bit/products/micro-bit-complete-starter-kit (UK)
micro:bit Motor Driver Board 	https://www.kitronik.co.uk/5620-motor-driver-board-for-the-bbc-microbit-v2.html (UK)
Batteriefach für Motor Driver Board 	https://www.kitronik.co.uk/c2235-4xaa-battery-cage-with-clip.html (UK) + Clip https://www.kitronik.co.uk/c2238-pp3-battery-clip-lead-heavy-duty.html (UK) Oder sonst nach 4xAAA Batteriefach suchen, in jedem Elektronik-Online-Shop erhältlich.
Calliope 	https://www.exp-tech.de/plattformen/sonstige/8229/calliope-mini (DE) https://www.conrad.ch/de/calliope-board-calliope-mini-1 (CH) https://educatec.ch/tectools/robotik-und-mikrokontroller/2142/calliope-mini-satz (CH)
Buzzer 	https://www.kitronik.co.uk/c3303-piezo-buzzer-no-drive.html (UK) Und sonst in jedem Elektronik-Online-Shop erhältlich.
Solenoid (linearer Motor) 	https://www.kitronik.co.uk/2540-solenoid-5v.html (UK) Und sonst in jedem Elektronik-Online-Shop erhältlich.
Potentiometer 	https://www.kitronik.co.uk/3010-10k-10k-16mm-linear-potentiometer.html (UK) Und sonst in jedem Elektronik-Online-Shop erhältlich.
LED gross 	https://www.kitronik.co.uk/3583-10-red-10mm-diffused-led-900mcd-pack-of-10.html Und sonst in jedem Elektronik-Online-Shop erhältlich.

Arcade Button 	https://www.kitronik.co.uk/catalogsearch/result/?cat=0&q=4677 (UK) In jedem Elektronik-Online-Shop erhältlich.
Krokodilklemmen 	https://www.kitronik.co.uk/2463-miniature-crocodile-clip-leads-pack-of-12.html (UK) Und sonst in jedem Elektronik-Online-Shop erhältlich.
Electric Paint	https://www.kitronik.co.uk/4804-bare-conductive-paint-10ml-pen.html (UK) Sonst auch in vielen anderen Online-Shops erhältlich
Kupfer-Klebeband	https://www.kitronik.co.uk/2477-copper-tape-with-conductive-adhesive-5mm-15m.html (UK)

6.2 Weitere Online-Shops

Online-Shops für alles rund um Elektronik und Roboterkits sind in Tabelle 10 aufgelistet. Alle Shops versenden die Ware in die Schweiz, dabei können Zollgebühren anfallen. Es lohnt sich, die Preise zu vergleichen!

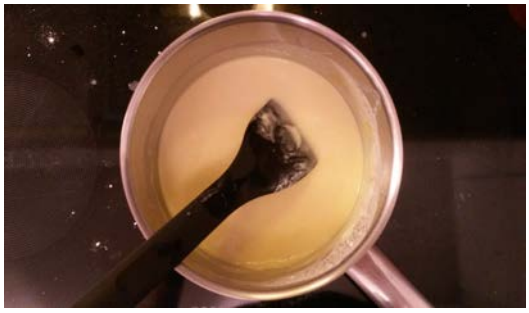
Tabelle 10 – Online-Shops rund um Elektronik.

Shop	**Ursprungsland**
https://educatec.ch/	CH
https://www.bastelgarage.ch/	CH
https://www.conrad.ch/	CH
https://www.distrelec.ch/	CH
https://www.mouser.ch/	DE
https://www.exp-tech.de	DE
http://ch.farnell.com/	
https://www.digikey.ch/	
https://www.kitronik.co.uk/	UK
https://shop.pimoroni.com/	UK
https://www.adafruit.com	USA
https://www.sparkfun.com/	USA
https://www.pololu.com/	USA
https://www.seeedstudio.com/	CHN

7 Zum Kochen

7.1 Leitfähiger Teig

120 ml Wasser
60 g Mehl
57 g Salz
1.5 EL Weinsteinsäure
0.5 EL ÖL
Einige Tropfen Lebensmittelfarbe



1. Alle Zutaten mischen.

2. Auf niedriger Hitze einkochen.

3. Einkochen, bis die Masse fest ist.

4. Etwas abkühlen lassen und kneten.

5. Das Rezept ergibt ca. 200 g Teig. Der Teig ist ungiftig, aber ungeniessbar.

6. Luftdicht verpacken. Der Teig ist dann mehrere Wochen haltbar.

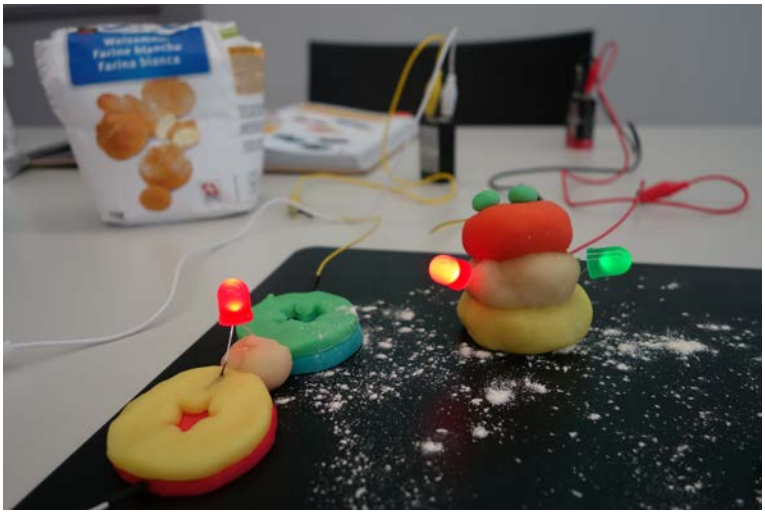
Der leitfähige Teig besitzt sehr viel Salz. Metallteile, die mit dem Teig in Kontakt kommen, korrodieren innert kürzester Zeit! Kontakte zum Schutz mit Alufolie umwickeln hilft.

7.2 Nicht-leitfähiger (isolierender) Teig

90 g Mehl
60 g Zucker
2 EL Öl
45 ml Destilliertes Wasser
Einige Tropfen Lebensmittelfarbe

1. Zutaten mischen und von Hand kneten.

2. Etwas Mehl dazugeben, bis der Teig nicht mehr klebrig ist. Das Rezept ergibt rund 260 g Teig, der aber nicht lange aufbewahrt werden kann!



Der isolierende Teig kann als Übergangsschicht zwischen dem negativ und positiv gepolten leitfähigen Teig verwendet werden. Der Teig ist ungiftig, aber ungeniessbar.