Impressum

1. Auflage 2016

0718021 AN 300416

Projektleitung: Jonathan Felder, Marc Gänssler, Sebastian Martin
Text und Versuche: Felix Homann
Produktentwicklung: Steffen Rothweiler
Gestaltungskonzept Anleitung: Atelier Bea Klenk, Berlin
Layout und Satz: Michael Schlegel, komuniki, Würzburg
Illustrationen: Michael Schlegel, komuniki, Würzburg
Fotos Anleitung: picsfive (alle Pinn-Nadeln); askaja (alle Büroklammern); Jaimie Duplass (alle Klebestreifen) (alle vorigen © fotolia.com); 8vFanl (S. 61 ur); hopsalka (S. 61 o); krista (S. 2 ul, S. 21 ur); tepic (S. 20 u) (alle vorigen © iStockphoto.com); Designua (S. 38 u) (© shutterstock.com); leejeongsoo (S. 35 u); theSOARnet (S. 38 o); Petr Kratochvil (S. 33 o) (alle vorigen © pixabay.com); Johannes (S. 2 mr, S. 21 or); Willi Heidelbach (S 27) (alle vorigen veröffentlicht unter CC BY-SA 2.5 einzusehen unter https://creativecommons.org/licenses/by-sa/2.5/deed.de); Philip Steffan (S. 60 alle, © Make Magazin); Alle Screenshots des Arduino-Programms © Arduino; Andreas Resch, argfx, St. Ulrich am Waasen (U1, S. 1 Rendering Gamepad, S. 1 und S. 32 SensorBots); Michael Flaig, proStudios, Stuttgart (S. 1 Zusatzmaterial); Matthias Kaiser, Stuttgart (alle anderen Fotos)

Gestaltungskonzept, Layout und Satz Verpackung: Michael Schlegel, komuniki, Würzburg
Fotos Verpackung: Andreas Resch, argfx, St. Ulrich am Waasen (alle Renderings); Michael Flaig, pro-studios, Stuttgart (Materialfoto)

Technische Änderungen vorbehalten. Printed in China / Imprimé en Chine

Art.-Nr.: 716032

EXPERIMENTIERKASTEN

Anleitung

GAMING • CODING
MAKING

KosmoBits

Spielend Programmieren lernen

Franckh-Kosmos Verlags-GmbH & Co. KG · Pfizerstraße 5 – 7 · 70184 Stuttgart · Telefon +49 (0) 711 2191-343

KOSMOS

1000 1001
1101 101 00
100 10
1010
0

› › › SICHERHEITSHINWEISE

Sicherheitshinweise

ACHTUNG! Nur für die Benutzung durch Kinder ab 10 Jahren oder älter. Anweisungen für Eltern oder andere verantwortliche Personen sind enthalten und müssen beachtet werden. Verpackung und Anleitung aufbewahren, da sie wichtige Informationen enthalten!

ACHTUNG! Nicht für Kinder unter 3 Jahren geeignet. Erstickungsgefahr, da kleine Teile verschluckt oder eingeatmet werden können.

Liebe Eltern,

dieser Experimentierkasten macht Ihr Kind auf spielerische Weise in spannenden Experimenten mit der Welt des Programmierens vertraut.
Bitte stehen Sie Ihrem Kind bei den Experimenten mit Rat und Tat zur Seite, unterstützen und begleiten Sie es.
Natürlich stellen Sie die Frage nach der Sicherheit. Dieser Kasten entspricht den Europäischen Sicherheitsnormen. Diese Normen enthalten Auflagen für den Hersteller, sehen aber auch vor, dass die Eltern ihren Kindern bei den Experimenten mit Rat und Tat zur Seite stehen.

Sagen Sie Ihrem Kind ausdrücklich, dass es alle relevanten Anweisungen und Sicherheitshinweise lesen und nachschlagebereit halten soll. Machen Sie es auch darauf aufmerksam, dass es die Hinweise und Regeln beim Experimentieren unbedingt einhalten muss.

Wir wünschen Ihrem Kind und natürlich auch Ihnen viel Spaß und Gewinn beim Experimentieren!

Hinweise zum Unweltschutz
Die elektronischen Teile dieses Produkts sind wiederverwertbar und dürfen zum Schutz der Umwelt am Ende ihrer Verwendung nicht über den Haushaltsabfall entsorgt werden. Sie müssen an einem Sammelpunkt für Elektroschrott abgegeben werden. Dieses Symbol weist darauf hin:
Bitte erfragen Sie bei Ihrer Gemeindeverwaltung die zuständige Entsorgungsstelle.

Entsorgung des Akkus
Der Akku gehört nicht in den Hausmüll! Jeder Verbraucher der EG ist gesetzlich verpflichtet Batterien und Akkus bei einer Sammelstelle seiner Gemeinde oder im Handel abzugeben. Der Akku wird dadurch einer umweltschonenden Entsorgung zugeführt. Batterien und Akkus, die Schadstoffe enthalten, sind durch dieses Zeichen oder durch chemische Symbole gekennzeichnet (Cd = Cadmium, Hg = Quecksilber, Pb = Blei).

Hinweise zum Umgang mit dem Akku
- Für die Experimente wird ein Lithium-Polymer-Akku verwendet. Bitte ausschließlich den mitgelieferten Akku verwenden!
- Der Akku darf nur unter Aufsicht von Erwachsenen geladen werden.
- Keine Experimente mit den Steckdosen des Lichtnetzes durchführen! Keine Drähte oder Bauteile in Steckdosen einführen! Die Netzspannung (230 Volt!) ist lebensgefährlich!
- Beim Experimentieren einen Kurzschluss des Akkus vermeiden, er könnte überhitzen und explodieren!
- Anschlüsse des Akkus dürfen nicht kurzgeschlossen werden!
- Der Akku muss mit der richtigen Polarität eingelegt werden.
- Verformungen des Akkus vermeiden.
- Bei Defekt des Akkus muss dieser aus dem Spielzeug herausgenommen werden.
- Bewahre das Set außerhalb der Reichweite von kleinen Kindern auf.
- Der Zusammenbau des Geräts und das Einsetzen des Akkus, muss entsprechend den Anweisungen in dieser Anleitung erfolgen, um die Zerstörung von Bauteilen zu vermeiden.

Hinweise zum Umgang mit den elektronischen Bauteilen
- Den Kontakt mit metallischen Gegenständen und Flüssigkeiten aller Art vermeiden!
- Nach dem Experimentieren alle empfindlichen Bauteile in die vorgesehenen Tüten verpacken und zusammen mit den anderen Teilen im Experimentkasten aufbewahren.
- Sollte die KosmoBits-Hardware längere Zeit nicht benutzt werden, bitte das Anschlusskabel des Akkus vom Interaction Board lösen.
- Das Spielzeug darf nur an Geräte der Schutzklasse II angeschlossen werden, die das folgende Bildzeichen tragen:

Vereinfachte EU-Konformitätserklärung
Hiermit erklärt die Franckh-Kosmos Verlags-GmbH & Co. KG, dass der Funkanlagentyp 620141 KosmoBits der Richtlinie 2014/53/EU entspricht.
Der vollständige Text der EU-Konformitätserklärung ist unter der folgenden Internetadresse verfügbar:
http://www.kosmos.de/content-943-943/entsorgung_und_sicherheit/

Für neugierige Forscher

Entdecken, verstehen, Spaß haben!

▸ Mondkrater, Galaxien, Planeten und Gasnebel – all das rückt zum Greifen nah, wenn du mit dem Linsenteleskop auf Entdeckungsreise gehst. Durch die hohe optische und technische Qualität macht das Beobachten besonders viel Spaß. Auch Vögel oder Waldtiere lassen sich mit der beiliegenden Bildumkehrlinse hervorragend beobachten.

ab 12 Jahren

ab 12 Jahren

▸ Mit dem KOSMOS-Qualitätsmikroskop kannst du nicht nur klassische mikroskopische Präparate wie das Zwiebelhäutchen im Durchlicht, sondern auch flache Objekte wie Blätter oder Münzen im Auflicht betrachten. Die hohe Qualität der optischen und mechanischen Teile ermöglichen dir spannende Untersuchungen in der Welt der kleinen Dinge. Mit vier Dauerpräparaten, umfangreichem Präparierzubehör sowie umfassender Anleitung mit vielen praktischen Tipps.

kosmos.de

SICHERES EXPERIMENTIEREN
KOSMOS
Qualität
SEIT 100 JAHREN

Sicherheit und Qualität:

KOSMOS Experimentierkästen werden von unserem erfahrenen Team mit größter Sorgfalt entwickelt und in allen Entwicklungs- und Produktionsschritten geprüft.

Im Hinblick auf die Produktsicherheit entsprechen unsere Experimentierkästen den europäischen Spielzeugrichtlinien und unseren eigenen durch langjährige Erfahrung entstandenen Standards. Um höchste Sicherheit zu gewährleisten, arbeiten wir bei unseren chemischen Experimentierkästen mit zertifizierten Prüfstellen zusammen.

Durch die enge Zusammenarbeit mit unseren Partnern in der Produktion sind wir in der Lage, alle Arbeitsschritte der Fertigung zu kontrollieren. Traditionell stellen wir die Mehrzahl unserer Produkte in Deutschland her, aber auch bei Experimentierkästen, die im Ausland produziert werden, gewährleisten wir damit die Einhaltung aller geforderten Standards.

Was in deinem Experimentierkasten steckt:

GUT ZU WISSEN! Die Teile des Kastens kannst du natürlich nachbestellen. Lade dir dazu einfach einen Bestellschein unter kosmos.de herunter.

Checkliste: Suchen – Anschauen – Abhaken

✔	Nr.	Bezeichnung	Anzahl	Art.-Nr.
❍	1	KosmoDuino	1	717 982
❍	2	Interaction Board	1	717 981
❍	3	Gamepad Gehäuse-Oberseite rechts	1	718 006
❍	4	Gamepad Gehäuse-Oberseite links	1	718 007
❍	5	Gamepad Gehäuse-Unterseite	1	718 005
❍	6	Rad mit Rückstellfeder	1	718 008 718 009
❍	7	Knöpfe mit Gummistempel	1	718 010 718 011
❍	8	Lichtsensor	1	717 985
❍	9	Schallsensor	1	717 986
❍	10	Temperatursensor	1	717 984
❍	11	Bewegungssensor	1	717 983
❍	12	Gehäuse für Schallsensor	1	718 000 718 004
❍	13	Gehäuse für Lichtsensor	1	717 999 718 003
❍	14	Gehäuse für Temperatursensor	1	717 997 718 001

✔	Nr.	Bezeichnung	Anzahl	Art.-Nr.
❍	15	Gehäuse für Bewegungssensor	1	717 998 718 002
❍	16	Breadboard	1	717 996
❍	17	Jumperkabel männlich-weiblich	10	717 990
❍	18	Jumperkabel männlich-männlich	10	717 989
❍	19	Widerstände: 330 Ohm	5	717 991
❍	20	LEDs: gelb grün blau rot	je 1	717 994 717 993 717 995 717 992
❍	21	Kabel: USB zu Micro-USB	1	717 988
❍	22	Lithium-Polymer-Akku 800 mAh, 3,7 V (ohne Bild)	1	717 987

Was du zusätzlich brauchst:

Smartphone/Tablet mit Android (ab 4.3) oder iOS (ab Version 7), das Gerät muss Bluetooth 4 oder höher unterstützen. PC mit Internetzugang

1000 1001
1101 101 00
100 10
1010
0

››› INHALT

Sicherheitshinweise ..... **Vordere Umschlaginnenseite**
Ausstattung **1**
Inhalt **2**

So startest du mit KosmoBits **3**
Das Gamepad und die Sensoren zusammenbauen
Die App

Die Welt der Mikrocontroller **6**

Vorbereitung **8**
Installation der Arduino-Software

Projekt 1: Blink! **9**
Ein Programm auf deinen KosmoDuino hochladen

Projekt 2: Ausschalter **14**

Projekt 3: Einschalter **15**

Das Interaction Board **17**

Projekt 4: Buntes Licht **18**

Projekt 5: Auf Tastendruck **22**

Projekt 6: Blink-Würfel **26**

Projekt 7: Der serielle Monitor **28**

Die for-Schleife **30**

Sensoren **32**

Projekt 8: Thermometer **33**

Projekt 9: Finger-Disko **36**

Projekt 10: Piep! **38**

Projekt 11: Zufallstöne! **39**

Projekt 12: Sirene **40**

Projekt 13: Tonleiter **43**

Projekt 14: Sensor-Orgel **44**

Projekt 15: Der serielle Plotter **46**

Projekt 16: Der Klatschschalter **47**

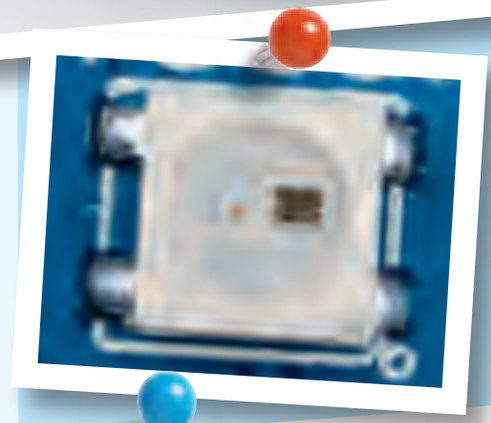
Projekt 17: Der Schubladenwächter **48**

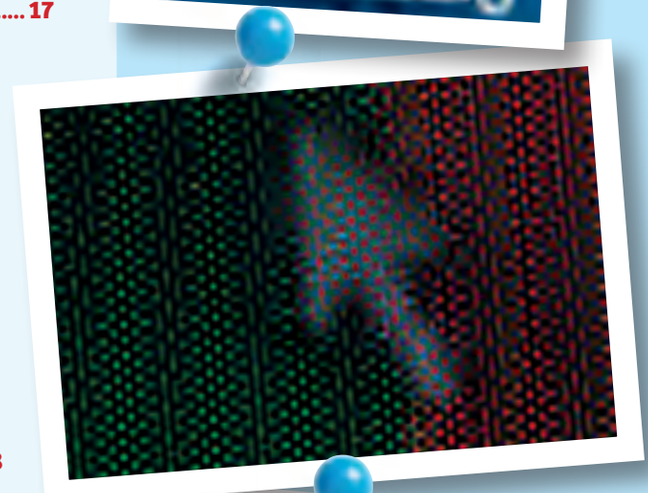
Projekt 18: In den Kühlschrank geschaut **53**

Projekt 19: Geisteraugen **55**

Häufige Fehlermeldungen **62**

Notizen **64**
Impressum ..... **Hintere Umschlagaußenseite**





TIPP!
Zusätzliches Wissen findest du auf den »Nachgehakt«-Seiten 21, 59–61 und den »Wissen Kompakt«-Seiten 24–25, 35, 41–42 und 58.

LOS GEHT'S

So startest du mit KosmoBits

DAS GAMEPAD ZUSAMMENBAUEN

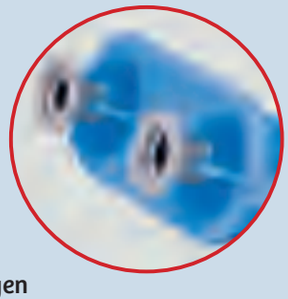
1. Zuallererst musst du die kleine Metall-Feder in die richtige Stelle im linken Teil der Gehäuse-Unterseite stecken. Die Feder hält später das Rad und sorgt dafür, dass es immer wieder in seine ursprüngliche Position zurückspringt.

2. Als nächstes musst du den dünnen Stift des Rades in das Loch des Dreh-Encoders stecken. Pass dabei auf, dass der Stift nicht abbricht. Der Dreh-Encoder ist auf dem Interaction Board befestigt und dort auch beschriftet.

3. Verbinde den Akku mit dem Interaction Board. Der Anschluss befindet sich auf der Unterseite. Der Stecker des Akkus ist so geformt, dass er sich nur mit der richtigen Polung verbinden lässt. Wende beim Verbinden keine Gewalt an!

4. Der Akku hat seinen festen Platz in der Gehäuse-Unterseite. Lege ihn so hinein, dass er fest sitzt und nicht wackelt.

5. Jetzt kannst du das Interaction Board in das Gehäuse einsetzen. Achte darauf, dass das Rad richtig in der Feder sitzt (siehe rundes Bild).



6. Die Knöpfe müssen auf der Unterseite mit den grauen Gummistempeln bestückt werden. Diese einfach mit der dünneren Seite in die Aussparungen auf der Unterseite der Knöpfe einstecken. Dann kannst du die Knöpfe von unten in die rechte Gehäuse-Oberseite einfügen.

   Anschließend musst du noch die restlichen Gehäuseteile aufstecken. Achte auf die richtige Positionierung der Knöpfe. Die Knöpfe sitzen richtig, wenn du beim Drücken einen klaren Druckpunkt spürst.

7. Dein Gamepad ist jetzt eigentlich fertig, es fehlt nur noch die »Schaltzentrale«, der KosmoDuino. Diesen kannst du einfach auf das Interaction Board aufstecken. Achte darauf, dass alle Pins passen und sich beim Einstecken nicht verbiegen. Drücke den KosmoDuino so weit rein, dass seine »Füßchen« nicht mehr zu sehen sind.

DIE SENSORBOTS ZUSAMMENBAUEN

1. Nimm einen der Sensoren und suche dir das entsprechende Gehäuse mit der richtigen Farbe (Bewegungssensor – blau; Temperatursensor – rot; Lichtsensor – lila; Schallsensor – gelbgrün. Siehe S. 1 »Ausstattung«). Stecke den Sensor mit der Vorderseite voran in die Gehäusehälfte mit den aufgedruckten Augen. Die »Füßchen« der Sensoren sind immer auf der Rückseite befestigt. Achte beim Temperatursensor darauf, dass der kleine schwarze Knubbel leicht aus dem Loch im Gehäuse ragt.

2. Jetzt musst du nur noch die Gehäuse-Rückseite mit der Gehäuse-Vorderseite verbinden. Drücke sie dafür fest zusammen.

SYSTEMVORAUSSETZUNGEN:

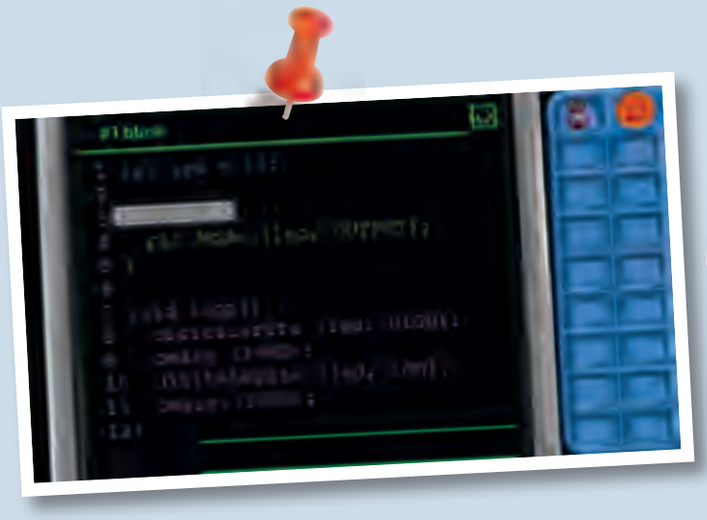
KosmoBits unterstützt Geräte ab den Betriebssystemen Android 4.3 und iOS 7 und neuer. Das Gerät muss Bluetooth 4 oder höher unterstützen.

DIE APP

Um dir den Einstieg in die Welt des Programmierens zu erleichtern, haben wir eine App entwickelt, mit der du erste Erfahrungen zu dem Thema sammeln kannst.

Kern der App ist ein Spiel, in dem du immer mal wieder kleine Programmier-Rätsel lösen musst. Aber keine Angst – diese sind nicht schwer, sodass du sie bestimmt alle knacken kannst. Der Schlüssel zum Lösen der Rätsel sind die Codemonster, die du in jedem Level findest. Sammle alle ein, denn du brauchst sie an den Computerterminals. Die Computerterminals enthalten lückenhaften Code. Diese Lücken füllst du, indem du das richtige Codemonster in die passende Lücke ziehst. Um mehr über die Monster zu erfahren, kannst du sie in deinem Inventar einfach oder doppelt antippen - so verraten sie, welche Lücke im Code sie ausfüllen können.

Um die Figuren im Spiel zu steuern, kannst du dein Gamepad verwenden. Dafür muss Bluetooth auf deinem Tablet oder Smartphone aktiviert sein. Starte einfach die App und schalte dein Gamepad ein, indem du den Schalter am oberen Rand auf die Position »ON« schiebst. Die Verbindung erfolgt dann automatisch. Ist die Verbindung aktiv, verschwinden die Steuerelemente auf dem Bildschirm und du kannst die App nur noch mit dem Gamepad steuern. Schaltest du dein Gamepad aus, erscheinen die Steuerelemente wieder. Ohne das Gamepad macht das Spiel aber nur halb so viel Spaß!



◄▲ *Verschiedene Screenshots der KosmoBits-App*

SO LÄDST DU DIE KOSTENLOSE APP HERUNTER:

Für das Installieren einer App benötigst du Zugang zu Google Play oder zum Apple App Store. Bitte deine Eltern um Unterstützung beim Installieren der App.

Wenn du ein Gerät mit Android hast, öffne Google Play, mit einem IPhone musst du den App Store öffnen. Gib einfach den Suchbegriff KosmoBits ein und installiere die App.

Laden im App Store

APP ERHÄLTLICH BEI Google Play

Für das Spiel benötigst du deine vier SensorBots. Die sind sehr nützlich und helfen dir, Hindernisse im Spiel aus dem Weg zu räumen. Du kannst sie einfach in dein Gamepad einstecken (siehe Bild). Sobald du einen Sensor eingesteckt hast, verwandelt sich dein Spiele-Charakter in einen der vier Bots. Jeder von ihnen hat seine ganz eigenen Fähigkeiten. Hast du z.B. den blauen Bot "Newton" eingesteckt, musst du dein Gamepad kräftig nach vorne und hinten schütteln, damit seine Fähigkeiten aktiviert werden. Du findest bestimmt selbst heraus, wie du bei den anderen Sensorbots die Spezialfähigkeiten aktivierst.

Spiel das Spiel in Ruhe durch und schau dir auch die anderen Inhalte der App im Hauptmenü an. Dort erfährst du schon eine Menge zu deinem Experimentierkasten und zum Thema Programmieren.

Viel Spaß dabei!

Die Welt der Mikrocontroller

Willkommen in der Welt der Mikrocontroller! Sie sieht deiner eigenen Welt erstaunlich ähnlich. Genauer gesagt: Sie sieht haargenau so aus wie deine! Überall in deiner Umgebung verstecken sich unzählige Mikrocontroller und verrichten unbemerkt ihre Arbeit. Kaum ein elektrisches Gerät funktioniert noch ohne einen Mikrocontroller: Die elektrische Zahnbürste, die Fernbedienung des Fernsehers, das Gamepad, das Digitalthermometer, die sprechende Puppe, der bellende Spielzeughund, die Waschmaschine, der Rauchmelder, der Toaster, und und und ...

▲ *KosmoBits-Controller*

ABER WAS IST EIN *MIKROCONTROLLER?*

Ein Mikrocontroller ist ein ziemlich kleiner (»Mikro«) Computer. Er unterscheidet sich allerdings grundlegend von den Computern, mit denen du normalerweise zu tun hast: Er hat keine Tastatur, keine Maus, auch einen Bildschirm findest du nicht.

Stattdessen besitzt er eine Menge kleiner Füßchen. Die werden *Pins* genannt. Ein Pin ist so etwas wie die Verbindung des Mikrocontrollers zur Außenwelt. Einige der Pins haben festgelegte Funktionen. Die meisten sind allerdings sogenannte **GPIO**-Pins. Das ist eine Abkürzung für »Allzweckeingabe/-ausgabe« (engl. **G**eneral **P**urpose **I**nput / **O**utput). Diese GPIO-Pins können zur Ein/- oder Ausgabe benutzt werden. So, wie du es gerade brauchst.

1. EINGABE:

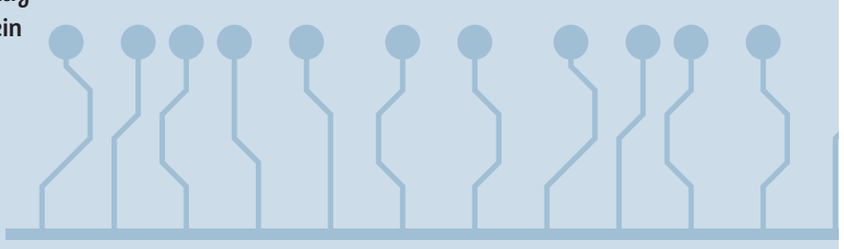
Du kannst einen Eingabe-Pin zum Beispiel mit einem Sensor verbinden. Dieser überwacht die Außenwelt und schickt über wechselnd hohe elektrische Spannung Informationen über die Umgebung, wie Temperatur oder Helligkeit an den Prozessor. Bei deinem Gamepad registriert ein Eingabe-Pin auch, wenn du eine Taste drückst.

2. AUSGABE:

Der Prozessor kann je nach Programmierung auf diese Informationen reagieren. Dafür brauchst du Ausgabe-Pins, die die Verbindung zu LEDs oder zum Soundmodul herstellen, welche durch die Höhe der weitergeleiteten elektrischen Spannung mit verschiedenen Tönen und Farben reagieren.

Richtig spannend wird es dann, wenn du mehrere Pins gleichzeitig benutzt. So kann der Mikrocontroller selbstständig auf die Sensoren reagieren, also beispielsweise Alarm schlagen, wenn es ihm zu warm wird.

Was dein Mikrocontroller mit einem normalen Computer gemein hat, ist der Hauptprozessor, das Gehirn eines jeden Computers. Das ist das kleine schwarze Viereck oben links auf dem Mikrocontroller. In der Mitte befindet sich ein kleinerer Prozessor. Dieser ist für die Kommunikation zwischen Micocontroller und PC verantwortlich. Rechts sitzt der Bluetooth-Chip, der eine drahtlose Verbindung ermöglicht.



WAS BEDEUTET EIGENTLICH ARDUINO?

In dieser Anleitung wird dir immer wieder das Wort »Arduino« begegnen. Das ist der Name für eine weit verbreitete Mikrocontroller-Plattform, also Platinen und dazugehörige Entwicklungsumgebung, von der der KosmoDuino abgeleitet wurde.

Lange Zeit war die Mikrocontroller-Programmierung nur etwas für Spezialisten. Die Begründer des Arduino-Projektes haben es sich im Jahr 2005 deshalb zur Aufgabe gemacht, die Programmierung und Nutzung einfacher Mikrocontroller so simpel wie möglich zu gestalten. Kunststudenten, die nie zuvor etwas mit Programmierung zu tun hatten, sollten damit in der Lage sein, Mikrocontroller in ihrer Arbeit zu verwenden.

*Rund um die Arduino-Plattform hat sich mittlerweile eine große **Community** gebildet, die unzählige interessanter Projekte entwickelt hat. Viele davon kannst du mit deinem KosmoDuino nachbauen: Er ist mit einem Arduino Uno Board kompatibel.*

Mehr zum Arduino-Projekt erfährst du auf www.arduino.cc. Allerdings ist die Seite englischsprachig.

DER KOSMOBITS-CONTROLLER

Mit deinem **KosmoBits-Controller** hältst du nun deinen eigenen Mikrocontroller in den Händen. Wir haben ihn KosmoDuino genannt, da er vom Arduino-Mikrocontroller abgeleitet wurde. Was er genau macht, ist deine Entscheidung. Denn damit er überhaupt etwas macht, muss er von dir *programmiert* werden.
Wie das geht, erfährst du in dieser Anleitung.

Vorbereitung

Um deinen KosmoDuino zu programmieren, benötigst du einen normalen PC oder einen Laptop mit der Arduino-Software. Die Software wird auch Programmier-Umgebung genannt, da sie eine Vielzahl von Hilfsmitteln zusammenfasst, die du zum Programmieren brauchst.

INSTALLATION DER KOSTENLOSEN ARDUINO-SOFTWARE

Die Arduino-Software lädtst du von der Seite **https://www.arduino.cc/en/Main/Software** herunter. Dort findest du verschiedene Versionen für die gängigsten Betriebssysteme. Für Windows-Betriebssyteme wählst du »Windows Installer« aus.

▲ *Downloadseite auf www.arduino.cc*

Auf der folgenden Seite wirst du um eine Spende gebeten. Das Arduino-Projekt, das die Entwicklung der Arduino-Software vorantreibt, finanziert sich zu einem großen Teil aus solchen Spenden. Du kannst die Software aber auch ohne Spende kostenlos herunterladen, indem du auf »Just Download« klickst.

Nach dem Download führst du die heruntergeladene Datei aus. Da die Arduino-Software ständig überarbeitet wird, gibt es immer wieder neue Versionen. Zu dem Zeitpunkt, als diese Anleitung entstanden ist, war die Version 1.6.6 aktuell, die Datei hieß *arduino-1.6.6-windows.exe*.
Wir empfehlen aber das Arbeiten mit der Version 1.6.5, da diese von uns ausgiebig getestet wurde. Ältere Versionen findest du auf der Seite unter »PREVIOUS RELEASES«.

Nach dem Start der Datei folgst du den Anweisungen des Installationsprogramms.

INSTALLATION DER KOSMOBITS-BIBLIOTHEKEN UND BEISPIELE

In einer sogenannten Software-Bibliothek werden nützliche und wieder verwendbare Funktionen zur Verfügung gestellt, die dir das Programmieren erleichtern. Wir haben die KosmoBits-Bibliotheken entwickelt. Sie enthalten auch sämtliche Programmier-Beispiele, die du in dieser Anleitung findest.

Die KosmoBits-Bibliotheken kannst du auf der Seite **www.kosmobits.de/downloads** herunterladen. Die Datei heißt **KosmoBits_Archiv.zip**.
Es handelt sich um eine .zip-Datei, die du in den Arduino-Ordner entpacken musst.

1. Öffne den Explorer.
2. Gehe in den Downloads-Ordner.
3. Öffne die Datei *KosmoBits_Archiv.zip* mit einem Doppelklick.
4. Markiere alle in dem Archiv vorhandenen Dateien und Ordner, indem du gleichzeitig die Tasten »Strg« und »A« drückst.
5. Drücke dann die Tasten »Strg« und »C«, um die ausgewählten Dateien zu kopieren.
6. Gehe in den Ordner *Dokumente → Arduino → libraries*.
7. Drücke nun gleichzeitig die Tasten »Strg« und »V«, um die Dateien und Ordner einzufügen.

Prima, jetzt kannst du mit dem Programmieren beginnen. Du brauchst nur noch die Arduino-Software zu starten, und schon kann es losgehen!

ERKLÄRUNGEN ZUM PROGRAMM-CODE:

Hier steht die Erklärung zum Programm-Code. Teile aus dem Code sind immer `orange` hinterlegt.

```
Hier steht der Programm-Code
// Kommentare zum Code sind immer grau.
// Dieser Text gehört nicht zum Code, sondern
// dient der Erklärung und der Übersichtlichkeit
```

Dein erstes Programm: Blink!

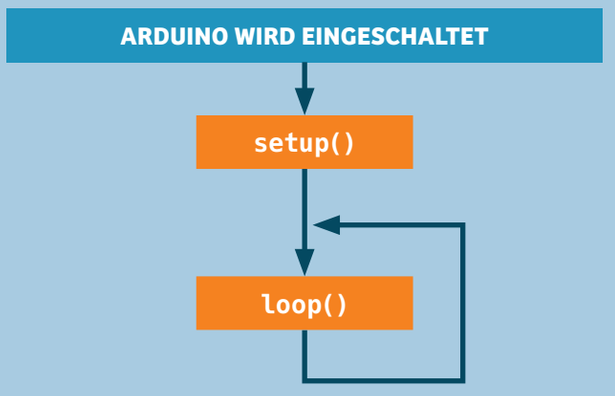
Um dein erstes Programm zu schreiben, startest du am PC die Arduino-Umgebung. Es öffnet sich ein Fenster, in das du deinen Programm-Code eingeben kannst. Ein paar Zeilen sind bereits für dich vorgefertigt worden:

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Das ist das Grundgerüst eines jeden Arduino-Programms:

1. Dein KosmoDuino arbeitet zunächst alle Anweisungen in der *Funktion* `setup()` ab, also alle Anweisungen, die in den geschweiften Klammern nach `setup()` stehen.
2. Danach wird die Funktion `loop()` aufgerufen. Das heißt, dein Mikrocontroller arbeitet alle Anweisungen innerhalb der geschweiften Klammern nach `loop()` ab. Wenn die letzte Anweisung abgearbeitet wurde, ruft dein Controller die Funktion `loop()` erneut auf. Das heißt, die Anweisungen in `loop()` werden in einer Endlosschleife immer und immer wieder aufgerufen. Daher auch der Name: *loop* ist Englisch und bedeutet *Schleife*.

Das kann man sich gut mit einem sogenannten *Flussdiagramm* veranschaulichen: Das Programm folgt dabei immer der Pfeilrichtung: Vom Einschalten, zu `setup()`, zu `loop()` und dann immer wieder zu `loop()` zurück.



▲ *Flussdiagramm*

Du könntest das bereits als Programm auf deinen KosmoDuino hochladen. Da aber weder in `setup()` noch in `loop()` irgendwelche Anweisungen stehen, würde das Programm eben auch nichts machen.

Deshalb gibst du deinem KosmoDuino jetzt etwas zu tun! Ändere dazu den Programm-Code folgendermaßen ab:

```
int ledPin = 13;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
}
```

Im Bereich der Arduino-Programmmierung wird für ein Programm auch oft der Begriff »Sketch« verwendet. In dieser Anleitung werden beide Begriffe benutzt.

Lass dich von den englischen Begriffen und unterschiedlichen Klammern nicht abschrecken! Wir gehen das Programm gleich Schritt für Schritt durch. Dann wirst du sehen, dass es gar nicht so schwer zu verstehen ist!
Vorher wollen wir uns aber ansehen, was das Programm eigentlich macht. Du musst es also auf deinen KosmoDuino laden.

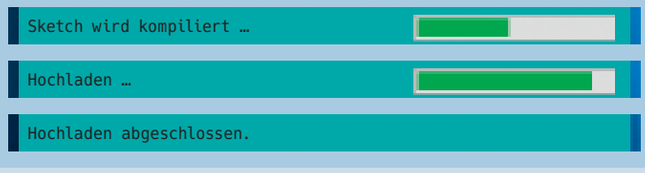
EIN PROGRAMM AUF DEN KOSMODUINO HOCHLADEN

Um einen Sketch auf deinen KosmoDuino hochzuladen, gehst du wie folgt vor:

1. Schließe den KosmoDuino mit dem USB-Kabel an deinen Computer an. Das Interaction Board benötigst du nicht.
2. Speichere deinen Programm-Code wenn nötig ab. Klicke dazu auf das »Speichern«-Symbol in der Arduino-Umgebung.
3. Stelle sicher, dass die richtige Platine ausgewählt ist: Werkzeuge → Platine → »Arduino/Genuino Uno«.
4. Stelle sicher, dass du den richtigen Port ausgewählt hast: Werkzeuge → Port → z.B. »COM3« (Arduino/Genuino Uno).
5. Klicke jetzt in der Arduino-Umgebung auf das »Hochladen«-Symbol.

Wenn du alles richtig gemacht hast, sollten in der Statuszeile der Arduino-Umgebung nacheinander die folgenden Meldungen erscheinen:

1. Sketch wird kompiliert ...
2. Hochladen ...
3. Hochladen abgeschlossen.



*Du kannst immer nur ein Programm gleichzeitig auf deinen KosmoDuino hochladen. Wenn du mit dem GamePad das KosmoBits-Spiel spielen möchtest, musst du die Datei **KosmoBits_App.ino** aufspielen. Diese Datei ist auch bei Auslieferung des Kastens auf dem KosmoDuino vorinstalliert.*

FEHLERMELDUNG?

Solltest du eine Fehlermeldung sehen, schaue dir deinen Text noch einmal genau an. Hast du vielleicht etwas falsch geschrieben? Die Stelle, an der ein Fehler im Programmtext zu Problemen führt, wird in der Arduino-Umgebung rot markiert, siehe Abbildung. Allerdings ist der wirkliche Fehler oft schon vorher zu suchen. Schon kleine Tippfehler führen in der Regel zu Fehlermeldungen! Solltest du keinen Fehler finden, dann schau dir den gelben Kasten weiter unten an. Dort wird unter »Beispiel-Programme öffnen« erklärt, wie du die in der Anleitung beschriebenen Beispiele nutzen kannst, ohne sie selbst eintippen zu müssen.

Tippfehler: »pinnMode« statt »pinMode«! Die Fehlerstelle ist rot hinterlegt. ►

Hat alles geklappt? Prima! Dann solltest du jetzt sehen, dass auf dem KosmoDuino eine kleine, grüne LED – die sogenannte »Onboard-LED« – regelmäßig blinkt.

Beispiel-Programme öffnen

Alle Programmbeispiele kannst du auch direkt öffnen. Du findest sie in der Arduino-Umgebung unter

»Datei« → »Beispiele« → »KosmoBits«.

ERKLÄRUNG

Jetzt wird es aber Zeit, das Programm zu verstehen. Schauen wir es uns doch einmal genauer an.

Die erste Zeile lautet: `int ledPin = 13;`

Anweisungen müssen in der Arduino-Programmiersprache mit einem Semikolon (Strichpunkt) abgeschlossen werden.

Hier wird eine *Variable* mit dem Namen `ledPin` definiert. Variablen sind wichtige Bestandteile eines jeden Programms. In Variablen kannst du nämlich Werte speichern – also Zahlen, Buchstaben oder ganze Wörter. Jede Variable hat einen Namen. Unter dem Namen kannst du den gespeicherten Wert wieder abrufen.

Wenn du einen Wert in einer Variablen speichern möchtest, benutzt du das Gleichheitszeichen »=«: Auf die linke Seite des Gleichheitszeichens kommt die Variable, auf die rechte

der Wert. `int ledPin = 13` bedeutet also, dass der Wert 13 in der Variablen `ledPin` gespeichert wird. Man sagt: »Der Variablen `ledPin` wird der Wert 13 zugewiesen.«

Wo immer du in deinem Programm die Zahl 13 benutzen möchtest, kannst du nun statt »13« auch »ledPin« schreiben.

WAS ABER BEDEUTET `int`?

In der Arduino-Programmiersprache kann eine Variable nicht beliebige Werte aufnehmen. Die Werte müssen von einem bestimmten ***Typ*** sein, den man festlegen muss. In diesem Fall ist der Typ `int`. Das bedeutet, dass die Variable sogenannte Integer-Werte aufnehmen kann. Das sind für deinen KosmoDuino die ganzen Zahlen von -32768 bis 32767.

```
void setup() {
  pinMode(ledPin, OUTPUT);
}
```

Hier wird eine **Funktion** definiert. Sie heißt `setup()`. Wenn eine Funktion im Programm-Code aufgerufen wird, werden alle in ihr enthaltenen Anweisungen Schritt für Schritt abgearbeitet. Das sind alle Anweisungen, die zwischen den geschweiften Klammern stehen. Eine Funktion kannst du dir also als ein kleines Unterprogramm vorstellen.

Die Funktion `setup()` ist übrigens eine spezielle Funktion. Sie wird bei jedem Start deines KosmoDuinos automatisch als erstes aufgerufen.

Was aber macht die Funktion `setup()`? Sie ruft eine andere Funktion auf: `pinMode()`. Mit dieser Funktion legst du fest, in welcher Betriebsart ein Pin arbeiten soll. In diesem Fall soll der Pin `ledPin` (Pin Nummer 13), als `OUTPUT`, also als Ausgabe-Pin, arbeiten.

Ein Ausgabe-Pin funktioniert wie ein Schalter, den du programmgesteuert ein- oder ausschaltest. Die anderen möglichen Betriebsarten lernst du in späteren Projekten kennen.

Wenn die Anweisungen in `setup()` von deinem KosmoDuino abgearbeitet worden sind, wird als nächstes die Funktion `loop()` aufgerufen. Schauen wir uns an, was diese Funktion beinhaltet:

```
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(500);
  digitalWrite(ledPin, LOW);
  delay(500);
}
```

FUNKTIONEN

Mit Funktionen kannst du deine Programme in kleinere Blöcke aufteilen. Das hilft, den Überblick zu behalten, und für Ordnung im Programm zu sorgen. Außerdem kannst du häufig benutzte Abfolgen von Anweisungen in eine Funktion verpacken. Statt jedesmal den gleichen Text im Programm-Code zu wiederholen, kannst du dann einfach die entsprechende Funktion aufrufen.

Ähnlich wie bei der Definition einer Variablen, muss man auch bei der Definition einer Funktion zunächst einen Typ angeben. In diesem Fall `void`.
Das ist der sogenannte ***Rückgabetyp****. Eine Funktion kann nämlich am Ende einen Wert an das aufrufende Programm zurück geben. Hier ist der Rückgabetyp* `void`*. Das ist Englisch und bedeutet »leer«. Mit anderen Worten: Diese Funktion liefert keinen Wert zurück! Ein Beispiel für die Rückgabe eines Wertes ist die Temperatur, die vom Sensor gemessen wurde.*

Wichtig: Gib deinen Funktionen immer einen Namen, der bereits ausdrückt, was die Funktion macht. Das hilft dabei, ein Programm zu verstehen, wenn man es liest.

Was passiert, wenn `loop()` aufgerufen wird? Die erste Anweisung in `loop()` ist:

```
digitalWrite(ledPin, HIGH);
```

Mit `digitalWrite()` (engl. für »digital Schreiben«) kannst du steuern, ob an einem Ausgabe-Pin eine Spannung anliegt oder nicht. Beim Aufruf musst du dafür der Funktion `digitalWrite()` zwei Dinge mitteilen:

1. Welcher Pin ist gemeint?
2. Soll die Spannung am Pin ein- (`HIGH`) oder aus- (`LOW`) geschaltet werden?

In diesem Fall wird also die Spannung an Pin Nummer 13 (`ledPin`) eingeschaltet: Die LED leuchtet!
Wenn du einen anderen Pin einschalten möchtest, also z. B. Pin Nummer 9, dann schreibst du `digitalWrite(9, HIGH);`.

Die nächste Anweisung ist:

```
delay(500);
```

»Delay« ist Englisch und bedeutet »Verzögern«. Mit der `delay()` Anweisung kannst du den Programmablauf für eine bestimmte Zeit verzögern, also einfach warten. Wie lange, das wird durch die Zahl bestimmt, die du beim Aufruf von `delay()` übergibst, hier `500`. Die Zeit wird in Millisekunden (Tausendstelsekunden) angegeben. 500 Millisekunden sind eine halbe Sekunde. Das Programm wartet eine halbe Sekunde lang und macht nichts.

Danach folgt die Anweisung `digitalWrite(ledPin, LOW);`.
Das kannst du vielleicht jetzt schon alleine verstehen. Richtig: Die LED wird wieder ausgeschaltet!

Danach wartet der KosmoDuino wieder eine halbe Sekunde: `delay(500);`.

Und dann?

Dann geht das Ganze wieder von vorne los! `loop()` ist nämlich eine besondere Funktion in der Arduino-Programmierung. Sie wird endlos wiederholt! Alle Befehle in `loop()` werden wie in einer Endlosschleife immer und immer wieder ausgeführt. Man nennt das die **Hauptschleife**. `loop()` ist somit die wichtigste Funktion in jedem KosmoBits-Programm. Hier wird gesteuert, was der KosmoDuino tatsächlich macht, während `setup()` die nötigen Vorarbeiten erledigt.

Was hier passiert, wenn die Hauptschleife wiederholt wird, hast du bereits gesehen: Im regelmäßigen Abstand von einer halben Sekunde wird die LED auf dem KosmoDuino ein- und wieder ausgeschaltet, und wieder ein und wieder aus, usw. Mit anderen Worten: Die LED blinkt.

TYPEN *Einige geläufige Typen und ihre Bedeutung*

TYP	BEDEUTUNG	WERTE
`int`	Ganze Zahlen oder HIGH / LOW (an / aus)	-32768 bis 32767
`long`	Ganze Zahlen, groß	-2 147 483 648 bis 2 147 483 647
`float`	Gleitkommazahlen: »Zahlen mit Komma«	z.B. 1.5; 3.141; 2.678 (engl. Schreibweise: Punkt statt Komma!)
`double`	Wie `float` aber mit doppelter Genauigkeit	z.B. 3.141964; 21.45873
`char`	Einzelne Buchstaben	z.B. `a`; `A`; `c`; `C`
`const`	Nicht veränderbare Größe	Kann alle Werte annehmen

PROJEKT 2

ACHTUNG! Nicht geeignet für Kinder unter 10 Jahren. Falls ein inkorrekter Kurzschluss durch unterschiedliche Polaritäten verursacht, oder der Kondensator unter falschen Bedingungen verwendet wird, können heiße Oberflächen durch Bauteile auf der Platine entstehen.

Ausschalter

In deinem ersten Sketch »Blink« hast du gelernt, wie du mit einem Programm eine LED auf dem KosmoDuino blinken lassen kannst. Dazu hast du einen Pin des Controllers als Ausgabe-Pin benutzt, und damit die LED ein- und ausgeschaltet.

Du kannst einen Pin aber auch als Eingabe-Pin benutzen. Es wird an dem Pin dann keine Spannung ein- oder ausgeschaltet. Stattdessen kannst du in deinem Programm feststellen, ob gerade eine elektrische Spannung an den Pin angelegt ist.

Das kannst du nutzen, um die LED mit einem einfachen Schalter ein- und auszuschalten.

DU BRAUCHST

- **KosmoDuino**
- **2 Jumperkabel männlich-männlich (Erklärung, siehe S. 16)**

VORBEREITUNG

Stecke das eine Jumperkabel auf Pin 9, das andere auf einen der Gnd-Pins. Gnd steht für Ground (engl. für Erdung). Diese Pins haben keine eigentliche Funktion, sondern leiten nur Strom ab.

DER PLAN

```
int ledPin = 13;
int schalterPin = 9;
int schalterWert = HIGH;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(schalterPin, INPUT_PULLUP);
}

void loop() {
  schalterWert = digitalRead(schalterPin);
  digitalWrite(ledPin, schalterWert);

  delay(50);
}
```

Lade den Sketch wie auf Seite 10 beschrieben auf deinen KosmoDuino. Nach dem Hochladen beginnt die LED zu leuchten. Wenn du jetzt die freien Kontakte der beiden Jumperkabel zusammenführst, so dass sie sich berühren, hört sie auf zu leuchten.

DAS PROGRAMM

Was also macht das Programm?

Du definierst zunächst einmal drei Variablen:

- `ledPin` für den Pin, an dem die LED angeschlossen ist.
- `schalterPin` weist du den Wert `9` zu, denn du schließt deinen »Schalter« an Pin 9 an.
- `schalterWert` bekommt erst einmal den Wert `HIGH`. Du wirst später Pin 9 »auslesen« und den gelesenen Wert in `schalterWert` speichern.

```
int ledPin = 13;
int schalterPin = 9;
int schalterWert = HIGH;
```

PROJEKT 3

ACHTUNG! Nicht geeignet für Kinder unter 10 Jahren. Falls ein inkorrekter Kurzschluss durch unterschiedliche Polaritäten verursacht, oder der Kondensator unter falschen Bedingungen verwendet wird, können heiße Oberflächen durch Bauteile auf der Platine entstehen.

Das ist fast wie in deinem ersten Programm. Der `ledPin` wird als Ausgabe-Pin (`OUTPUT`) betrieben. Pin 9 (`schalterPin`) wird allerdings als Eingabe-Pin betrieben. Deshalb benutzt du in der `pinMode()`-Anweisung für `schalterPin` nicht den Wert `OUTPUT`, sondern `INPUT_PULLUP`.

```
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(schalterPin, INPUT_PULLUP);
}
```

Zunächst liest du mit `digitalRead(schalterPin)` den aktuellen Zustand von Pin 9 aus. Ist der Pin elektrisch mit einem »Gnd« Pin verbunden, so ergibt das den Wert `LOW`. Das bedeutet es liegt keine Spannung an. Andernfalls den Wert `HIGH`. Diesen Wert speicherst du in der Variablen `schalterWert`.
Anschließend schreibst du diesen Wert in den LED Pin. Wenn Pin 9 also mit dem Gnd-Pin verbunden ist, schaltest du die LED aus. Andernfalls schaltest du sie ein.

```
void loop() {
  schalterWert = digitalRead(schalterPin);
  digitalWrite(ledPin, schalterWert);

  delay(50);
}
```

Danach wartest du mit `delay(50)` kurz, bevor das Ganze wiederholt wird.

Einschalter

DU BRAUCHST

- **KosmoDuino**
- **2 Jumperkabel männlich-männlich**

VORBEREITUNG

Stecke das eine Jumperkabel auf Pin 9, das andere auf einen der Gnd-Pins.

Lade den Sketch wie auf Seite 10 beschrieben, auf deinen KosmoDuino. Jetzt bleibt die LED nach dem Hochladen dunkel. Nur wenn sich die beiden freien Kontakte der Jumperkabel berühren, leuchtet sie.

DER PLAN

Im vorherigen Projekt hast du gelernt, wie du eine LED mit Hilfe eines einfachen »Schalters« ausschalten kannst. Was aber machst du, wenn es genau umgekehrt funktionieren soll, die LED also erst angeht, wenn der Kontakt geschlossen wird? Du änderst das Programm!

DAS PROGRAMM

```
int ledPin = 13;
int schalterPin = 9;
int schalterWert = HIGH;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(schalterPin, INPUT_PULLUP);
}

void loop() {
  schalterWert = digitalRead(schalterPin);

  if(schalterWert == LOW) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
  delay(50);
}
```

ERKLÄRUNG

Wir schauen uns nur die `loop()`-Funktion an, denn nur dort hat sich im Vergleich zum »Ausschalter« etwas geändert.

Genau wie beim »Ausschalter« liest du als erstes den `schalterPin` aus und speicherst den aktuellen Wert, also `HIGH` oder `LOW` in der Variablen `schalterWert` ab.

Du möchtest jetzt aber nicht den gelesenen Wert in den LED-Pin schreiben, sondern den genau umgekehrten: Wenn `schalterWert` den Wert `LOW` hat, möchtest du die LED einschalten (`digitalWrite(ledPin, HIGH)`), sonst möchtest du die LED ausschalten (`digitalWrite(ledPin, LOW)`).

Das drückst du in deinem Programmtext mit `if(...){...} else{...}` aus. »if« ist Englisch für »wenn« und »else« bedeutet »sonst«. Du ahnst es schon: Damit kannst du im Programm auf Bedingungen reagieren.

Die Bedingung, auf die hier reagiert wird, ist `schalterWert == LOW`. Das kannst du als Frage verstehen: »Ist schalterWert gleich LOW?« Die Antwort darauf wertest du mit der `if()` Anweisung aus: Wenn `schalterWert` gleich `LOW` ist (`if(schalterWert == LOW)`), wird der Programm-Code in der ersten geschweiften Klammer ausgeführt: `digitalWrite(ledPin, HIGH)`. Die LED wird eingeschaltet.

Sonst (`else`) wird der Teil in der geschweiften Klammer ausgeführt: `digitalWrite(ledPin, LOW)`. Die LED wird ausgeschaltet.

Schließlich wartest du mit `delay(50)` kurz, bevor das Ganze wiederholt wird.

```
void loop() {
  schalterWert = digitalRead(schalterPin);

  if(schalterWert == LOW) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }

  delay(50);
}
```

JUMPERKABEL

Jumperkabel werden benutzt, um auf einfache Weise elektronische Bauteile miteinander zu verbinden. In diesem Experimentierkasten findest du zwei verschiedene Ausführungen: ***männlich-männlich*** *und* **männlich-weiblich**. *Das heißt wirklich so! »männlich-männlich« bedeutet, dass sich an beiden Seiten ein Stecker befindet. Bei »männlich-weiblich« befindet sich hingegen an einer Seite ein Stecker und an der anderen eine Buchse. Jumperkabel werden oft auch als Patchkabel bezeichnet.*

»Weibliches Ende«

»Männliches Ende«

Das Interaction Board

Zugegeben, die kleine LED, die du bislang zum Leuchten gebracht hast, ist nicht sehr aufregend. Trotzdem war sie hilfreich, um dir einen schnellen Einstieg in die Programmierung deines KosmoDuino zu ermöglichen.

Damit du ebenso schnell auch spannendere Dinge mit deinem Mikrocontroller machen kannst, haben wir das **Interaction Board** entwickelt. Du hast es vermutlich schon im Spielemodus kennen gelernt. Darauf befinden sich einige praktische elektronische Bauteile, die du in vielen Projektideen einsetzen kannst:

- Zwei Taster (»Button 1« und »Button 2«)
- Eine mehrfarbige LED (»NeoPixel«)
- Ein Drehrad (»Drehencoder«)
- Ein Lautsprecher (»Buzzer«)

Und ganz wichtig:

- Eine Buchse zum Anschluss der KosmoBits-Sensoren (Temperatur, Bewegung, Helligkeit, Lautstärke)
- Ein Akku, damit du deinen KosmoDuino auch ohne angeschlossenen Computer benutzen kannst.

Um mit dem Interaction Board zu arbeiten, musst du zunächst deinen KosmoDuino in die Sockel auf dem Interaction Board stecken, siehe Abbildung.

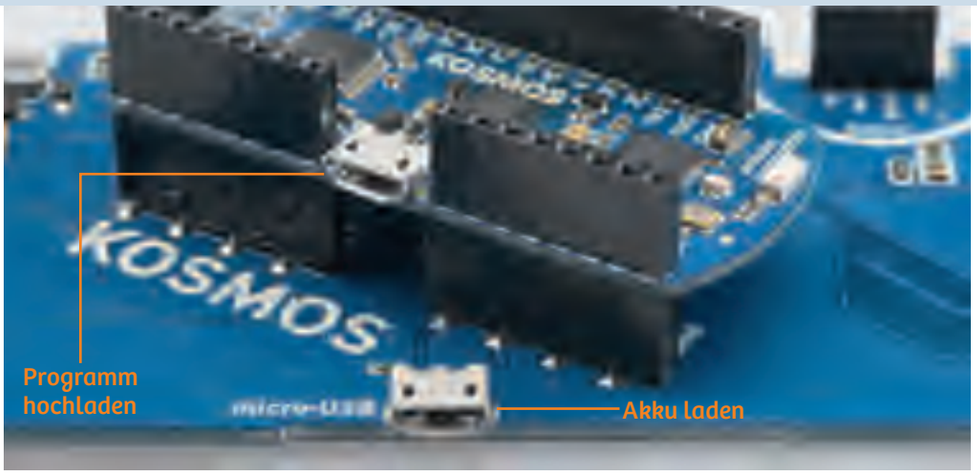
1

2

Dann kann es auch schon direkt los gehen! In den folgenden Projekten wirst du zunächst die Möglichkeiten des Interaction Boards erkunden, bevor es dann mit aufwendigeren Projekten weitergeht.

ACHTUNG!

Um ein Programm auf den Mikrocontroller hochzuladen musst du das USB-Kabel in die USB-Buchse des KosmoDuinos stecken. Der USB-Anschluss des Interaction Boards ist nur zum Laden des Akkus geeignet.



PROJEKT 4

Buntes Licht

DU BRAUCHST

› **KosmoDuino im Interaction Board**

DER PLAN

Auf dem Interaction Board findest du eine mehrfarbige LED, NeoPixel genannt. Damit kannst du die Farbe und die Helligkeit der LED selbst bestimmen. Wie das geht, lernst du in diesem kleinen Projekt, mit dem du den NeoPixel in unterschiedlichen Farben leuchten lässt.

DAS PROGRAMM

Das ist neu für dich. Mit einer `#include`-Anweisung, kannst du Programm-Code aus anderen Dateien in dein Programm einbinden. Hier wird der Inhalt aus den Dateien »Adafruit_NeoPixel.h« und »KosmoBits_Pixel.h« eingebunden.

Diese beiden Zeilen musst du immer dann am Anfang deines Programms einfügen, wenn du den NeoPixel benutzen möchtest!

```
#include <Adafruit_NeoPixel.h>
#include <KosmoBits_Pixel.h>
```

Bislang hast du nur Variablen kennen gelernt, die als Behälter für Zahlenwerte dienen. Die Variable `pixel`, die du hier definierst, ist anders: Sie nimmt keine Zahlen auf, sondern dient als Behälter für ein **Objekt** vom Typ *KosmoBits_Pixel*. Das klingt schwieriger, als es ist. Objekte können nur etwas mehr als einfache Werte wie `int`.
Sie haben nämlich nicht nur einen Wert, sondern bringen auch noch eigene Funktionen mit. Die werden auch als Methoden bezeichnet. Dein `pixel` zum Beispiel ist ein Objekt, mit dem du den NeoPixel auf dem Interaction Board ansteuern kannst. Dafür besitzt er eine Methode namens `setColor()`, mit der du Farbe und Helligkeit des NeoPixels einstellen kannst. Wie du sie benutzt, lernst du gleich in der Hauptschleife.

```
KosmoBits_Pixel pixel;
```

Um Farbe und Helligkeit deines NeoPixels einzustellen, benötigst du vier Zahlenwerte. Einen für die Helligkeit und drei für die Grundfarben rot, grün und blau. Die Werte speicherst du in den entsprechenden Variablen `rot`, `gruen`, `blau` und `helligkeit`.

Die unterschiedlichen Farben werden nämlich durch Mischen dieser drei Grundfarben erzeugt. Wie stark die jeweilige Farbe in dieser Mischung vertreten sein soll, stellst du mit dem zugehörigen Zahlenwert ein. Du kannst Werte von 0 bis 255 benutzen. 0 bedeutet, dass die jeweilige Farbe gar nicht leuchtet. Bei dem Wert 255 leuchtet die jeweilige Farbe mit höchstmöglicher Helligkeit.

```
int rot = 0;
int gruen = 0;
int blau = 0;
int helligkeit = 0;
```

In `setup()` ist diesmal nichts zu tun, da das Programm, das du mit der `#include`-Anweisung eingebunden hast, das schon für dich erledigt.

```
void setup() {
  // Hier ist nichts zu tun.
}
```

In der Hauptschleife wird zunächst der Helligkeitswert gesetzt: `helligkeit = 50;`. Die höchste Helligkeit wird mit dem Wert 255 erzielt, bei 0 bleibt der NeoPixel dunkel.

Danach werden der Reihe nach verschiedene Farben erzeugt. Dazu werden jeweils den Variablen `rot`, `gruen` und `blau` unterschiedliche Werte zugewiesen.

Entscheidend ist für jede Farbe die Zeile `pixel.setColor(rot, gruen, blau, helligkeit);`. Erst damit wird dem NeoPixel nämlich mitgeteilt, dass er eine neue Farbe darstellen soll! `setColor()` ist dabei eine sogenannte Methode des Objekts `pixel`. Um eine Methode eines Objekts aufzurufen, wird der Methodenname mit einem Punkt getrennt an den Namen des Objekts angehängt.

Mit `delay(500);` lässt du deinen KosmoDuino jeweils eine halbe Sekunde warten, bevor er die nächste Farbe einstellt.

NeoPixel leuchtet in unterschiedlichen Farben ▶

```
void loop() {
  // 50 ist ein guter Helligkeitswert.
  // Die höchste Helligkeitsstufe ist 255.
  // Das blendet aber schon sehr!
  helligkeit = 50;

  // rot
  rot = 255;
  gruen = 0;
  blau = 0;
  pixel.setColor(rot, gruen, blau, helligkeit);
  delay(500);

  // grün
  rot = 0;
  gruen = 255;
  blau = 0;
  pixel.setColor(rot, gruen, blau, helligkeit);
  delay(500);

  // blau
  rot = 0;
  gruen = 0;
  blau = 255;
  pixel.setColor(rot, gruen, blau, helligkeit);
  delay(500);

  // lila
  rot = 255;
  gruen = 0;
  blau = 255;
  pixel.setColor(rot, gruen, blau, helligkeit);
  delay(500);

  // türkis
  rot = 0;
  gruen = 255;
  blau = 255;
  pixel.setColor(rot, gruen, blau, helligkeit);
  delay(500);

  // gelb
  rot = 255;
  gruen = 255;
  blau = 0;
  pixel.setColor(rot, gruen, blau, helligkeit);
  delay(500);
```

```
  // weiß
  rot = 255;
  gruen = 255;
  blau  = 255;
  pixel.setColor(rot, gruen, blau, helligkeit);
  delay(500);
}
```

!

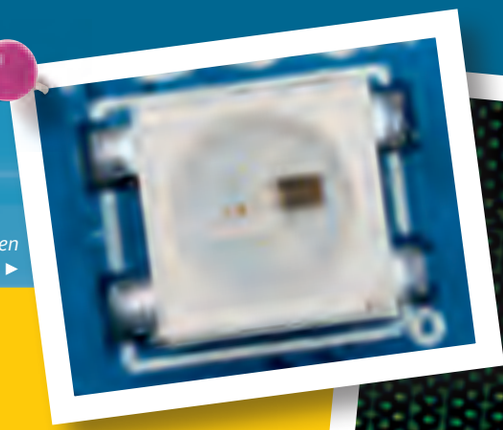
▲ Bibliothek

SOFTWARE-BIBLIOTHEKEN

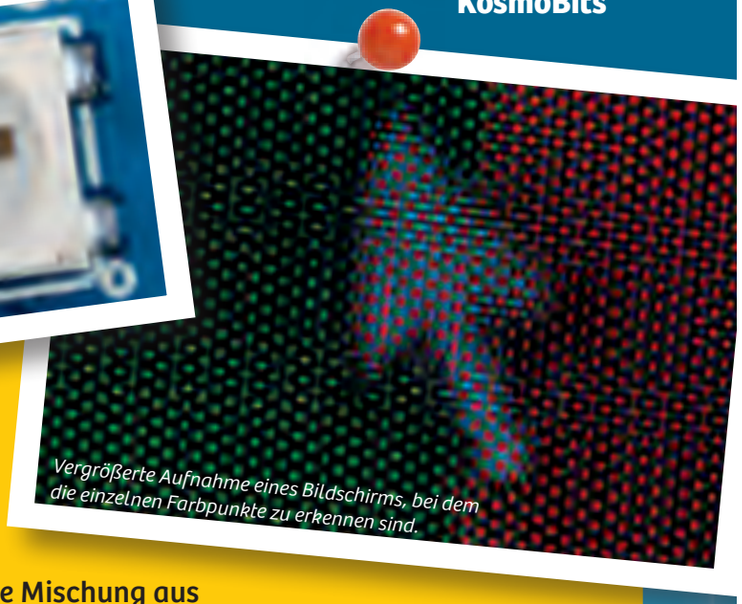
*Damit man nicht in jedem Programm das Rad wieder neu erfinden muss, gibt es sogenannte **Software-Bibliotheken**. In denen werden keine Bücher gesammelt, sondern wieder verwendbare Programmfunktionen und Definitionen. Es gibt Bibliotheken, die mathematische Funktionen zur Verfügung stellen, andere kümmern sich darum, dass man bestimmte Hardware, also zum Beispiel Sensoren oder auch den NeoPixel, komfortabel benutzen kann. Damit man die Mittel, die einem durch eine solche Bibliothek zur Verfügung gestellt werden, auch nutzen kann, muss man die jeweilige Bibliothek in das eigene Programm **einbinden**.*

Das geschieht mit Hilfe der `#include` *-Anweisung:* `#include <KosmoBits_Pixel.h>` *bindet zum Beispiel die KosmoBits_Pixel-Bibliothek ein. Manche Bibliothek verwendet aber auch Funktionen aus anderen Bibliotheken. Die muss man dann zusätzlich einbinden, wie im Beispiel der KosmoBits_Pixel-Bibliothek. Die benötigt nämlich zum Funktionieren die Adafruit_NeoPixel Bibliothek, die durch* `#include <Adafruit_NeoPixel.h>` *eingebunden wird.*

NACHGEHAKT



In der Nahaufnahme sind die drei einzelnen LEDs des NeoPixels gut zu erkennen. ►



Vergrößerte Aufnahme eines Bildschirms, bei dem die einzelnen Farbpunkte zu erkennen sind.

Additive Farbmischung

Was du als weißes Licht kennst, ist in Wirklichkeit eine Mischung aus Licht unterschiedlicher Farben. Aus welchen Farben es tatsächlich besteht, kannst du an einem Regenbogen ablesen. Ein Regenbogen entsteht nämlich dadurch, dass das Licht der Sonne durch die Regentropfen in der Luft in seine Einzelfarben aufgespalten wird.

Umgekehrt kannst du aber auch Licht unterschiedlicher Farben zusammen mischen. Das Ergebnis ist eine andere Farbe. Jeder Farbbildschirm nutzt dieses Prinzip: Der Fernseher, dein Computer, dein Smartphone oder Tablet. Bei all diesen Geräten besteht jeder einzelne Bildpunkt (»Pixel«) aus drei einfarbigen Bildpunkten in den Farben Rot, Grün und Blau. Weil sie so dicht beieinander liegen, kannst du sie aus der normalen Entfernung nicht einzeln wahrnehmen. Erst wenn du ganz dicht herangehst, oder eine Lupe benutzt, kannst du die einzelnen Farbpunkte erkennen.

Wenn du dir den NeoPixel auf dem Interaction Board genau anschaust, kannst du, solange er nicht leuchtet, drei verschiedene Bereiche erkennen. Jeder Bereich kann nur eine einzige Farbe erzeugen. Eben Rot, Grün oder Blau. Mit Hilfe der Methode `setColor()` sagst du dem NeoPixel, wie hell die einzelnen Bereiche leuchten sollen.

Versuche doch mal weitere Farben zusammen zu mischen, indem du die einzelnen Farbwerte variierst. Einen Anhaltspunkt, welche Mischungen welche Farben ergeben, liefert dir das Bild neben der Tabelle. Deine Ergebnisse kannst du in die folgende Tabelle eintragen:

▲ *In einem Regenbogen kannst du die einzelnen Farben des Sonnenlichts erkennen.*

Beschreibung der Farbe	Wert Rot	Wert Grün	Wert Blau
Rot	255	0	0
Grün	0	255	0
Blau	0	0	255

PROJEKT 5

Auf Tastendruck

In diesem Projekt wirst du lernen, wie du die Tasten auf dem Interaction Board benutzen kannst, um den NeoPixel des Interaction Boards in verschiedenen Farben leuchten zu lassen.

DU BRAUCHST

› **KosmoDuino im Interaction Board**

▲ *Finger drückt auf rechte Taste – NeoPixel leuchtet rot*

▲ *Finger drückt auf linke Taste – NeoPixel leuchtet blau*

DER PLAN

Der NeoPixel soll zunächst dunkel bleiben. Drückst du Taste 1 leuchtet der NeoPixel rot, drückst du Taste 2 leuchtet er blau. Wenn du beide Taste gleichzeitig drückst, leuchtet er lila.

DAS PROGRAMM

```
#include <Adafruit_NeoPixel.h>
#include <KosmoBits_Pixel.h>
#include <KosmoBits_Pins.h>
```

Wie man Bibliotheken und den Code aus anderen Dateien in ein Projekt einbindet, hast du bereits im letzten Projekt gelernt. Neu ist hier `KosmoBits_Pins.h`. Darin sind Namen für die verschiedenen vom Interaction Board genutzten Pins deines KosmoDuinos definiert. Die KosmoBits-Bibiliotheken werden wir uns später noch genauer anschauen.

▲ *Zwei Finger drücken auf beide Tasten – NeoPixel leuchtet lila*

Hier definierst du zwei ***Konstanten*** für die Pins, an die die beiden Taster des Interaction Boards angeschlossen sind. Diese werden mit `const` abgekürzt. Dieses Schlüsselwort teilt dem Programm mit, dass ein Objekt oder eine Variable nicht veränderbar ist.

```
const int taste1 = KOSMOBITS_TASTE_1_PIN;
const int taste2 = KOSMOBITS_TASTE_2_PIN;
```

Wie im vorigen Programm speicherst du die verschiedenen Farbwerte und die Helligkeit in entsprechenden Variablen, die du hier definierst.

```
int rot = 0;
int gruen = 0;
int blau = 0;
const int helligkeit = 50;
```

Um den NeoPixel auf dem Interaction Board steuern zu können, brauchst du wieder ein KosmoBits_Pixel Objekt. Das definierst du hier.

```
KosmoBits_Pixel pixel;
```

In `setup()` stellst du mittels `pinMode()` den Betriebsmodus für die Taster-Pins ein. Da die Taster über einen Widerstand an die Pins angeschlossen sind, benutzt du die Betriebsart `INPUT`.

```
void setup() {
  pinMode(taste1, INPUT);
  pinMode(taste2, INPUT);
}
```

In der ersten `if`-Anweisung wird mittels `digitalRead()` der Pin ausgelesen, an den Taste 1 angeschlossen ist. Wenn die Taste gedrückt ist, ergibt das den Wert `LOW`: `rot` wird der Wert `255` zugewiesen, also der größtmögliche Rotwert.
Wird die Taste nicht gedrückt, wird `rot` auf `0` gesetzt.

Das Ganze wird dann für Taste 2 in der zweiten `if`-Anweisung wiederholt. Nur, dass dort der Wert für `blau` geändert wird.

Mit `pixel.setColor(rot, gruen, blau, helligkeit);` wird schließlich der neue Farbwert des NeoPixels eingestellt.

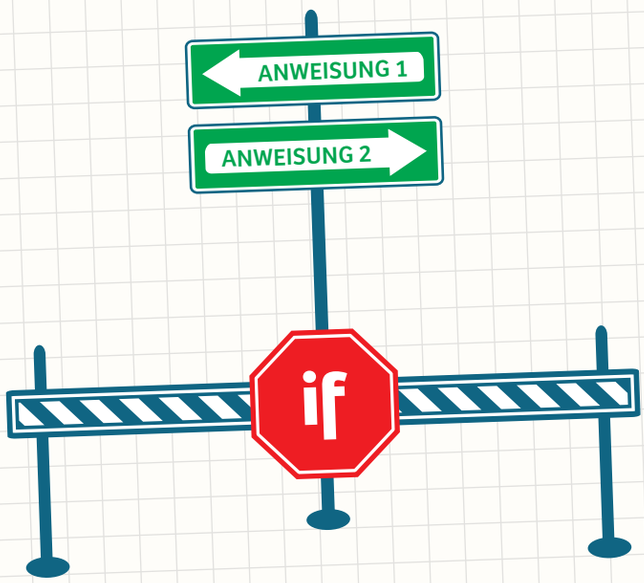
Am Ende der Hauptschleife lässt du den Controller noch 50 Millisekunden warten.

```
void loop() {

  if (digitalRead(taste1) == LOW) {
    rot = 255;
  } else {
    rot = 0;
  }

  if (digitalRead(taste2) == LOW) {
    blau = 255;
  } else {
    blau = 0;
  }
  pixel.setColor(rot, gruen, blau, helligkeit);

  delay(50);
}
```

WAS PASSIERT EIGENTLICH BEIM HOCHLADEN?

*Wenn du ein Programm auf deinen KosmoDuino hochlädst, passiert tatsächlich eine ganze Menge. So wie du es geschrieben hast, versteht der Mikrocontroller das Programm nämlich nicht: Er spricht die Programmiersprache einfach nicht. Deshalb muss das Programm erst in **Maschinensprache** übersetzt werden. Das macht ein sogenannter **Compiler** (gesprochen: »Kompeiler«).*

Die Programmiersprache, die du benutzt, ist also nur eine Zwischensprache, die du recht einfach verstehen und schreiben kannst, die aber vor allem der Compiler leicht in Maschinensprache übersetzen kann.



Nur unter einer Bedingung: Die if-Anweisung

Im Programm-Code hast du die `if`- (englisch für »wenn«) Anweisung bereits benutzt. Hier wollen wir sie uns noch etwas genauer anschauen. Mit Hilfe der `if`-Anweisung kannst du Teile deines Programms nur dann ausführen lassen, wenn eine bestimmte Bedingung erfüllt ist:

```
// Einfache if-Anweisung
if (Bedingung) {
  // Wird nur ausgeführt, wenn die
  // Bedingung erfüllt ist.
  Anweisung1;
  Anweisung2;
  ...
}
```

Die Programmteile in den geschweiften Klammern werden nur dann ausgeführt, wenn die Bedingung erfüllt ist. Andernfalls wird der gesamte Block in den gescheiften Klammern einfach übersprungen.

Die Bedingung muss einen **Wahrheitswert** zurück liefern: `true` (wahr) oder `false` (falsch). Meistens vergleicht man zwei Zahlenwerte. Dazu gibt es die **Vergleichsoperatoren**, die dir zumindest so ähnlich aus dem Mathematikunterricht bekannt sind:

OPERATOR	BEDEUTUNG	BEISPIEL	`true` (WAHR)
==	gleich	a == b	wenn a gleich b ist
<	kleiner als	a < b	wenn a kleiner als b ist
>	größer als	a > b	wenn a größer als b ist
!=	ungleich	a != b	wenn a ungleich b ist
<=	kleiner/gleich	a <= b	wenn a kleiner oder gleich b ist
>=	größer/gleich	a >= b	wenn a größer oder gleich b ist

Die Wahrheitswerte werden auch boolesche Typen genannt, abgekürzt mit `bool`.
Dieser kann lediglich die Werte true oder false annehmen.

ACHTUNG!

Ein häufiger und nicht immer leicht zu findender Fehler ist, in einer `if`-Anweisung, statt des doppelten Gleichheitszeichens, das einfache zu verwenden, also zum Beispiel statt `if (a == b)` die falsche Anweisung `if (a = b)`.
In dem Fall wird in der `if`-Anweisung `a` der Wert von `b` zugewiesen. Wenn dieser größer als Null ist, gilt das als wahr (`true`), und der Block in den geschweiften Klammern wird ausgeführt.

Du kannst eine `if`-Anweisung auch um einen `else`-Block erweitern. "Else" ist Englisch und bedeutet "sonst". Der `else`-Block wird also dann, und nur dann, ausgeführt, wenn die Bedingung **nicht** erfüllt ist.

```
// if-Anweisung mit else-Block
if (Bedingung) {
  // Wird nur ausgeführt, wenn die
  // Bedingung erfüllt ist.
  Anweisung1;
  Anweisung2;
  ...
} else {
  // Dieser Block wird ausgeführt, wenn
  // die Bedingung nicht erfüllt ist.
  Anweisung3;
  Anweisung4;
}
```

Schließlich kannst du mit `else if` (engl. »sonst, wenn«) noch einen Block einbauen, der eine zweite Bedingung überprüft, wenn die erste nicht erfüllt wurde.

```
// if-Anweisung mit else-Block
if (Bedingung1) {
  // Wird nur ausgeführt, wenn die
  // Bedingung erfüllt ist.
  Anweisung1;
  Anweisung2;
  ...
} else if (Bedingung2){
  // Dieser Block wird ausgeführt, wenn
  // die Bedingung nicht erfüllt ist.
  Anweisung3;
  Anweisung4;
} else {
  // Dieser Block wird ausgeführt, wenn
  // keine der vorher geprüften
  // Bedingungen erfüllt wird.
  Anweisung5;
  ...
}
```

Den abschließenden `else`-Block kannst du dabei auch weg lassen. Auch kannst du beliebig viele `else if`-Blöcke aneinander reihen.

PROJEKT 6

Blink-Würfel

Mit diesem Projekt machst du deinen KosmoDuino zu einem einfachen Würfel. Auf Tastendruck ermittelt er eine Zufallszahl von 1 bis 6 und signalisiert dir das Ergebnis durch Blinken des NeoPixels.

DU BRAUCHST

› **KosmoDuino im Interaction Board**

DER PLAN:

- Ist der Controller bereit, so leuchtet der NeoPixel blau.
- Drückst du auf Taste 1, beginnt der NeoPixel entsprechend des Würfelergebnisses grün zu blinken, also einmal, wenn eine Eins gewürfelt wurde, zweimal, wenn eine Zwei gewürfelt wurde, usw.
- Nach einer kurzen Pause leuchtet der NeoPixel wieder blau. Das bedeutet, er ist bereit erneut zu würfeln.

DAS PROGRAMM

Du bindest zunächst wieder die bekannten Bibliotheken ein und legst ein KosmoBits_Pixel an, damit du den NeoPixel auf dem Interaction Board ansprechen kannst.

Danach legst du ein paar Konstanten an, die den Code lesbarer machen.

```
#include <KosmoBits_Pins.h>
#include <Adafruit_NeoPixel.h>
#include <KosmoBits_Pixel.h>

KosmoBits_Pixel pixel;

const int tasterPin = KOSMOBITS_TASTE_1_PIN;
const int blinkDauer = 500; // Millisekunden
const int blinkPause = 250; // Millisekunden
const int helligkeit = 50;
```

In `setup()` bereitest du wie üblich deinen Controller auf die kommenden Aufgaben vor:
Mit `pinMode(tasterPin, INPUT)` stellst du den Pin, an den Taster 1 angeschlossen ist, als Input-Pin ein.

Später wirst du eine Zufallszahl erzeugen lassen. Das macht ein Zufallsgenerator, der aber einen möglichst zufälligen Startwert benötigt. Um den zu bekommen, liest du mit `analogRead(12)` Pin 12 aus. Da dort nichts angeschlossen ist, sind die Werte, die du so erhältst schon sehr zufällig. Die Funktion `analogRead()` wirst du später noch genauer kennenlernen. Mit `randomSeed(...)` wird der ausgelesene Wert dann dem eigentlichen Zufallsgenerator als Startwert übergeben (»random« bedeutet übersetzt zufällig).

Schließlich lässt du den NeoPixel mit `pixel.setColor(0, 0, 255, helligkeit);` blau leuchten.

```
void setup() {
  pinMode(tasterPin, INPUT);
  randomSeed(analogRead(12)); // Startwert für
  // Zufallsgenerator
  pixel.setColor(0, 0, 255, helligkeit); // Blau
  // signalisiert Betriebsbereitschaft
}
```

*Dies ist eine verkürzte Version der Schreibweise, wie du sie in Projekt 4 kennengelernt hast. Die Zahlen in den Klammern steuern den **RGB-Wert** (Rot/Grün/Blau).*

In der Hauptschleife `loop()` liest du mit `digitalRead()` den `tasterPin` aus. Wird er gedrückt, dann ist das Ergebnis `LOW`. Mit Hilfe der `if`-Anweisung wird dann die Funktion `wuerfeln()` aufgerufen, in der das Würfelergebnis ermittelt und ausgegeben wird. Andernfalls macht `loop()` nichts.

```
void loop() {
  if (digitalRead(tasterPin) == LOW) {
    wuerfeln();
 }
}
```

PROJEKT 6

Hier definierst du dir eine eigene Funktion namens `wuerfeln()`. Darin wird eine Zufallszahl im Bereich von 1 bis 6 ermittelt und anschließend ausgegeben. Gehen wir sie Schritt für Schritt durch:

Zunächst (1) machst du das Licht des NeoPixels aus und wartest kurz. Die entsprechenden Befehle kannst du mittlerweile sicherlich erkennen.

Jetzt geht es um das eigentliche Würfeln. Da dein KosmoDuino nicht wirklich würfeln kann, musst du anders an das Würfelergebnis kommen: Du befragst einen *Zufallsgenerator*.

Das machst du mit `random(1, 7)` (2). Damit fragst du den Zufallsgenerator nach einer zufällig ausgewählten Zahl. Der erste Parameter, hier 1, gibt an, dass die kleinste mögliche Zahl 1 sein soll. Der zweite Parameter, hier 7, bedeutet, dass die zufällig ausgewählte Zahl kleiner als 7 sein soll.

Wichtig: Der zweite Parameter muss also immer um 1 größer sein, als die größtmögliche Zahl, die vom Zufallsgenerator erzeugt werden soll. Das ist vielleicht etwas verwirrend, ist aber leider so. Die so erzeugte Zufallszahl wird in der Variablen `zahl` gespeichert.

Jetzt soll die Zufallszahl durch Blinken ausgegeben werden (3). Dazu bedienst du dich einer sogenannten `for`-Schleife. Den Aufbau der `for`-Schleife wirst du auf den folgenden Seiten ausführlich kennen lernen. An dieser Stelle ist erst einmal nur wichtig, dass der Code in den geschweiften Klammern genau `zahl`-mal ausgeführt wird. Wenn `zahl` den Wert 1 hat, 1 mal. Wenn `zahl` den Wert 2 hat, 2 mal usw.

Was geschieht in den geschweiften Klammern? Genau: Der NeoPixel leuchtet einmal für die Dauer `blinkDauer` grün, und wird dann für die Dauer `blinkPause` ausgeschaltet. Kurz: Der NeoPixel blinkt einmal grün. Da die `for`-Schleife aber `zahl`-mal durchlaufen wird, blinkt der NeoPixel insgesamt eben `zahl`-mal. Es wird also das Würfelergebnis durch Blinken ausgegeben.

Schließlich (4) wird nach einer weiteren kurzen Pause (`delay(blinkDauer)`) der NeoPixel wieder auf blau gestellt, um anzuzeigen, dass wieder gewürfelt werden kann.

Da `wuerfeln()` in deinem Programm nur aus der Hauptschleife `loop()` heraus aufgerufen wird, geht es anschließend zurück in die Hauptschleife. Als nächstes wird daher in `loop()` wieder geprüft, ob die Taste gedrückt ist oder nicht.

```
void wuerfeln() {
  // (1)
  // Erst einmal Licht aus, um zu zeigen, dass
  // auf den Tastendruck reagiert wird.
  pixel.setColor(0, 0, 0, 0); // Licht aus
  delay(500);

  // (2) Erzeuge Zufallszahl
  int zahl = random(1, 7);

  // (3) Blinke zahl-mal
  for (int i = 0; i < zahl; ++i) {
    pixel.setColor(0, 255, 0, helligkeit); // Grün
    delay(blinkDauer);
    pixel.setColor(0, 0, 0, 0); // Licht aus
    delay(blinkPause);
  }

  // (4) Abschluss des Würfelns signalisieren.
  delay(blinkDauer); // Pause am Ende etwas
  // länger
  pixel.setColor(0, 0, 255, helligkeit); // Blau
  // signalisiert Betriebsbereitschaft
}
```

Viel Spaß beim Würfeln!

Mehr Infos über die `for`-Schleife bekommst du auf den Seiten 30/31.

PROJEKT 7

Der serielle Monitor

DU BRAUCHST

› **KosmoDuino**
› **Computer**
› **USB-Kabel**

DER PLAN

Mit Hilfe des seriellen Monitors kannst du dir von deinem KosmoDuino kleine Botschaften auf den Bildschirm deines Computers schicken.

Um den seriellen Monitor nutzen zu können, gehst du folgendermaßen vor:

DAS PROGRAMM

Füge in `setup()` die Zeile `Serial.begin(115200);` ein.

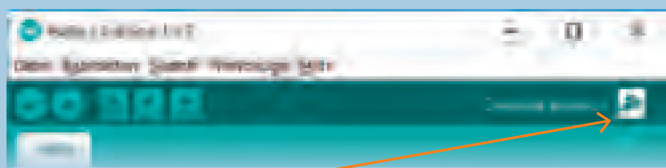
Wann immer du etwas auf den seriellen Monitor ausgeben möchtest, kannst du das nun mit Hilfe der Befehle `Serial.print()` und `Serial.println()` tun. Der Unterschied zwischen den beiden Varianten ist, dass `Serial.println()` noch einen Zeilenumbruch einfügt. Die nächste Ausgabe beginnt also in einer neuen Zeile.

Schauen wir uns das mit einem kleinen Beispiel an:

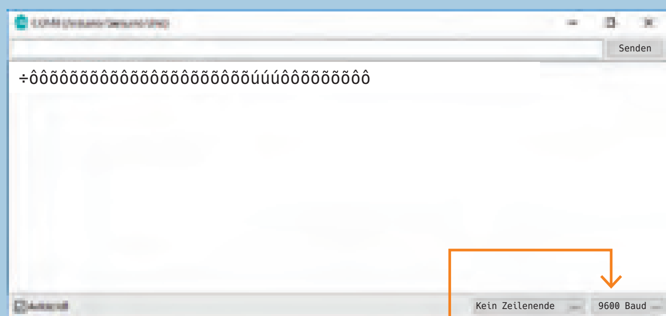
```
void setup() {
  Serial.begin(115200);
}

void loop() {
  Serial.print("Hallo!");
  delay(500);
}
```

Lade das Programm auf deinen KosmoDuino hoch. Scheinbar geschieht überhaupt nichts. Klicke jetzt auf das Lupensymbol oben rechts in der Arduino-Umgebung. Es öffnet sich ein neues Fenster, der serielle Monitor.

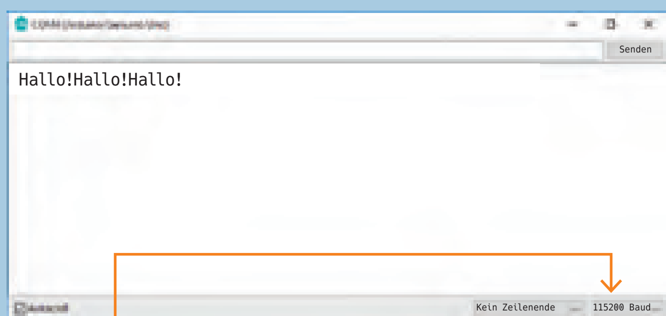
Vermutlich siehst du zunächst nur komische Zeichen. Das kannst du aber schnell ändern, indem du die richtige Übertragungsgeschwindigkeit einstellst. Unten rechts im seriellen Monitor findest du eine Auswahlliste, in der du »115200 Baud« auswählen musst. Dann solltest du sehen, wie ein »Hallo!« nach dem anderen in das Fenster geschrieben wird.



▲ *Mit einem Klick auf die Lupe oben rechts in der Arduino-Umgebung startest du den seriellen Monitor.*



▲ *Wenn die falsche Übertragungsrate, hier 9600 Baud, eingestellt ist, siehst du nur wirre Zeichen.*



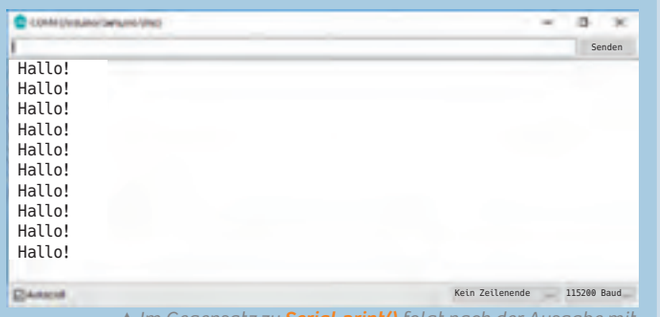
▲ *Ist die richtige Übertragungsgeschwindigkeit eingestellt, kannst du im seriellen Monitor »Botschaften« vom KosmoDuino empfangen.*

PROJEKT 7

Schließe jetzt den seriellen Monitor. Ersetze in der Hauptschleife den Aufruf `Serial.print("Hallo!");` durch `Serial.println("Hallo!");`:

```
void loop() {
  Serial.println("Hallo!");
  delay(500);
}
```

Lade das geänderte Programm auf den Controller und starte den seriellen Monitor erneut. Jedes »Hallo!« bekommt nun eine eigene Zeile.



▲ *Im Gegensatz zu **Serial.print()** folgt nach der Ausgabe mit **Serial.println()** immer eine neue Zeile.*

Mit `Serial.println()` kannst du aber nicht nur vorher festgelegte Texte ausgeben, sondern auch gerade erst ermittelte Messwerte. Das wirst du im nächsten Projekt, »Thermometer« auf Seite 33 kennen lernen.

ACHTUNG!

Wenn du versuchst, ein Programm auf den KosmoDuino zu laden, während der serielle Monitor geöffnet ist, kommt es manchmal zu einer Fehlermeldung. Schließe dann den seriellen Monitor und versuche, das Programm erneut hochzuladen.

Die for-Schleife

Wenn Programmteile wiederholt ausgeführt werden, spricht man von einer Schleife. Eine wichtige Schleife hast du bereits kennengelernt: Die Hauptschleife `loop()`. Darin erledigt dein KosmoDuino die wichtigsten Aufgaben deiner Programme. Er liest Eingabe-Pins aus, verarbeitet die Messdaten und reagiert schließlich darauf. Immer und immer wieder von vorne.

Oftmals möchtest du aber nicht in einer Endlosschleife gefangen sein, sondern wünschst dir nur eine bestimmte Anzahl von Wiederholungen. Dafür kannst du eine `for`-Schleife verwenden. Du wirst sie meistens in der folgenden Weise als sogenannte *Zählschleife* verwenden:

```
for (int i = 0; i < 20; ++i) {
  Anweisung1;
  Anweisung2;
  ...
}
```

Was bedeutet das? Mit dem Wort `for` wird die Schleife eingeleitet. Die Anweisungen, die wiederholt ausgeführt werden, stehen in dem Block in den geschweiften Klammern. Man nennt diesen Block mit den zu wiederholenden Anweisungen auch den *Schleifenrumpf*. Damit die Schleife aber auch weiß, wie oft diese Anweisungen wiederholt werden sollen, braucht die `for`-Schleife noch ein paar Angaben, die du in den runden Klammern finden kannst. Schauen wir uns die einmal genauer an:

`int i = 0;`:
Das ist die sogenannte Initialisierung der Schleife. Bevor die Schleife das erste Mal durchlaufen wird, legst du hier die *Zählvariable* `i` an und gibst ihr den Wert 0. Du kannst der Zählvariablen einen beliebigen Namen und Anfangswert geben. Solange die Zählvariable keine besondere Bedeutung hat, benutzt man meist die Buchstaben `i`, `j` und `k` als Namen. In aller Regel startet man mit dem Anfangswert 0.

`i < 20;`:
Das ist die Testbedingung: **Vor jedem Durchlauf** der Schleife wird diese Bedingung geprüft. Ist die Bedingung erfüllt, wird die Schleife durchlaufen, also der gesamte Programmcode in den geschweiften Klammern ausgeführt. Ist die Bedingung nicht erfüllt, wird die Schleife verlassen: Der Programmcode in den geschweiften Klammern wird kein weiteres Mal ausgeführt. Stattdessen wird das Programm mit der nächsten Anweisung nach der Schleife fortgesetzt.

`++i`:
Das ist zunächst einmal eine Kurzform für `i = i + 1`. Der Wert der Zählvariablen `i` wird also um 1 erhöht. Aber wann? **Nach jedem Schleifendurchlauf!**

Sofern im Schleifenrumpf der Wert der Zählvariablen `i` nicht verändert wird, passiert also folgendes:

Vor dem ersten Durchlauf hat `i` den Wert 0. Das ist kleiner als 20. Die Testbedingung ist also erfüllt, und die Schleife wird durchlaufen. Nachdem die Anweisungen im Schleifenrumpf ausgeführt wurden, wird nun der Wert der Zählvariablen um 1 erhöht. Der Wert von `i` ist jetzt 1. Die Bedingung ist immer noch erfüllt, der Schleifenrumpf wird ausgeführt, `i` wieder um 1 erhöht und so weiter und so weiter. Wenn `i` schließlich den Wert 19 hat, wird noch einmal der Schleifenrumpf ausgeführt und danach `i` auf den Wert 20 erhöht. Die Testbedingung ist dann nicht mehr erfüllt, die Schleife wird verlassen.

Wie oft ist der Schleifenrumpf jetzt ausgeführt worden? Er ist ausgeführt worden für die folgenden Werte der Zählvariablen `i`: 0, 1, 2, 3, ..., 19. Die Zahlen 1 bis 19 ergeben 19 Druchläufe, dazu kommt noch ein Durchlauf für den Wert 0. Es sind also insgesamt 20 Durchläufe!

Die Zählvariable ist übrigens nicht allein zum Zählen da. Ihren aktuellen Wert kannst du auch jederzeit im Schleifenrumpf verwenden. Wenn du zum Beispiel alle Zahlen von 1 bis 42 auf dem seriellen Monitor ausgeben möchtest, kannst du das so machen:

```
// Gibt die Zahlen 1, 2, 3,..., 42 aus:
for (int i = 1 ; i <= 42; ++i) {
  Serial.println(i); // Gibt den aktuellen
  // Wert von i aus
}
```

DIE FOR-SCHLEIFE

Hier startet die Schleife mit dem Anfangswert 1. Getestet wird diesmal nicht, ob `i` kleiner als 42 ist, sondern ob `i` kleiner oder gleich 42 ist. Das heißt, dass die Schleife auch für den Wert 42 noch einmal durchlaufen wird.

Es geht aber auch in umgekehrter Reihenfolge:

```
// Gibt die Zahlen 42, 41, 40, ..., 1 aus:
for (int i = 42; i >= 1; --i) {
  Serial.println(i);
}
```

Die Schleife startet mit dem Anfangswert 42. Sie wird solange durchlaufen, wie `i` einen Wert hat, der größer oder gleich 1 ist. Mit `--i` wird diesmal `i` nach jedem Schleifendurchlauf um 1 verringert, also rückwärts gezählt.

Tatsächlich kannst du anstelle von `++i` oder `--i` beliebige Anweisungen verwenden. Du kannst zum Beispiel nur die geraden Zahlen ausgeben:

```
// Gibt die Zahlen 2, 4, 6, ..., 42 aus.
  for (int i = 2; i <= 42; i = i + 2) {
  Serial.println(i);
}
```

Hier startet die Schleife bei 2 und nach jedem Schleifendurchlauf wird die Zählvariable mit `i = i + 2` gleich um 2 erhöht.

Sensoren

Bislang hast du gelernt, wie dein KosmoDuino auf einen Tastendruck reagieren kann. In diesem Experimentierkasten findest du aber auch eine Menge Sensoren, mit denen dein Mikrocontroller auf seine Umwelt reagieren kann: Die KosmoBits-Module!

Mit dem Temperatursensor kann dein Controller die Temperatur überwachen, mit dem Lichtsensor die Helligkeit. Der Bewegungssensor erlaubt es dir, auch leichteste Bewegungen des KosmoDuinos zu registrieren, und mit dem Schallsensor kannst du auf Geräusche warten.

Aber wie kommt der KosmoDuino an die Daten der Sensoren? Wie erfährt er, wie warm es ist, oder wie hell?

Ganz einfach: Die Sensoren geben ihre Messwerte in Form einer elektrischen Spannung an den Mikrocontroller weiter. Der kann diese Spannung dann an einem Pin auslesen. Viele der Pins können nämlich nicht nur »an« und »aus« unterscheiden, sondern auch unterschiedliche Spannungen messen. Dazu dient der Befehl `analogRead()`, mit dem du die elektrische Spannung an einem Pin bestimmen kannst. Das Ergebnis ist ein Zahlenwert von 0 bis 1023. Je höher die elektrische Spannung ist, die an dem Pin anliegt, umso höher ist der Wert. Bei 0 liegt gar keine Spannung an, bei 1023 sind es 5 Volt.

Natürlich musst du `analogRead()` auch sagen, welcher Pin ausgelesen werden soll. In der Regel wird das der Pin `KOSMOBITS_SENSOR_PIN` sein, der in der Datei `KosmoBits_Pins.h` definiert wird. Diese Datei solltest du also immer mit `#include <KosmoBits_Pins.h>` einbinden, wenn du auf die Sensoren der KosmoBits-Module zugreifen möchtest.

Den seriellen Monitor hast du bereits kennen gelernt. Das ist ein wichtiges Werkzeug, wenn du mit den Sensoren arbeiten möchtest. Warum, erfährst du in den nächsten Projekten.

PROJEKT 8

Thermometer

DU BRAUCHST

› **KosmoDuino im Interaction Board**
› **Temperatursensor**

VORBEREITUNG

Stecke den Temperatursensor in das Interaction Board.

DER PLAN

In diesem Projekt wirst du deinen KosmoDuino zu einem Thermometer machen. Die Temperatur wird dabei mit Hilfe des Temperatursensors gemessen. Die Ausgabe der Temperatur erfolgt über den seriellen Monitor.

DAS PROGRAMM

Schreibe jetzt den folgenden Code und lade ihn auf den Controller:

Das Programm ist schnell erklärt. Mit `analogRead()` wird in der Hauptschleife wiederholt die Spannung am `sensorPin` ausgelesen. Der Messwert wird dann über den seriellen Monitor ausgegeben. Du wirst dieses Programm immer wieder benutzen, wenn du einen neuen Sensor kennenlernen möchtest.

Öffne also den seriellen Monitor und schau dir die Messwerte an. Lege einen Finger auf den schwarzen Punkt, der aus dem Gehäuse des Sensors heraus schaut. Du solltest sehen, dass sich die gemessenen Werte verändern. Nach einer Temperatur sieht das aber noch nicht aus. Du musst also noch die Messwerte in eine Temperatur umrechnen. Dabei hilft dir die KosmoBits-Bibliothek. Darin findest du nämlich eine Funktion, die genau das macht. Sie heißt `thermistor_messwert_in_celsius()`.
Um sie nutzen zu können, musst du zuvor die Datei `KosmoBits_Thermistor.h` einbinden.

```
#include <KosmoBits_Pins.h>

const int sensorPin = KOSMOBITS_SENSOR_PIN;

void setup() {
  pinMode(sensorPin, INPUT);
  Serial.begin(115200);
}

void loop() {
  int wert = analogRead(sensorPin);

  Serial.print("Messwert: ");
  Serial.println(wert);
  delay(1000);
}
```

◄ *Board mit Temperatursensor*

ÄNDERE ALSO DEIN PROGRAMM SO AB:

Schließe den seriellen Monitor, lade das Programm hoch und öffne wieder den seriellen Monitor. Du siehst, dass nun in regelmäßigen Abständen die aktuelle Temperatur ausgegeben wird.

```
#include <KosmoBits_Pins.h>
#include <KosmoBits_Thermistor.h>

const int sensorPin = KOSMOBITS_SENSOR_PIN;

void setup() {
  pinMode(sensorPin, INPUT);

  Serial.begin(115200); // Ausgabe über seriellen
// Monitor ermöglichen.
}

void loop() {
  int messwert = analogRead(sensorPin);
  float celsius = thermistor_messwert_in_
celsius(messwert); // Messwert in Grad Celsius
// umrechnen.
  Serial.print(celsius); // Wert von celsius wird
// ausgegeben.
  Serial.println("Grad Celsius"); // Ausgabe von
// "Grad Celsius" und Zeilenumbruch.

  delay(1000);
}
```

WISSEN KOMPAKT

RECHNEN IM PROGRAMM:

Zahlen und Variable, die Zahlen aufnehmen können, hast du bereits kennen gelernt. Die nützen dir aber nicht viel, wenn du mit ihnen nicht rechnen kannst. Denn in aller Regel wirst du mit deinem KosmoDuino einen Zahlenwert aus einem Sensor auslesen, und damit dann etwas anderes steuern. Dazu musst du aus dem Messwert einen anderen Zahlenwert errechnen.

In der Arduino-Programmiersprache stehen dir dafür alle Grundrechenarten zur Verfügung. Für Plus- und Minusrechnung werden dabei die üblichen Symbole »+« und »-« benutzt. Für die Multiplikation benutzt man einen Stern »*«, für die Division den einfachen Schrägstrich »/«.

```
// Addition (plus)
int x; // x ist eine ganze Zahl.
x = 1 + 4; // x hat den Wert 5.

// Subtraktion (minus)
x = 5 - 3; // x hat den Wert 2.

// Multiplikation (mal)
x = 3 * 4; // x hat den Wert 12.

// Division (geteilt durch)
x = 12 / 6; // x hat den Wert 2.

// Punktrechnung vor Strichrechnung
x = 4 * 3 + 2; // x hat den Wert 14.
```

Du kannst übrigens auch den bisherigen Wert einer Variablen benutzen, um daraus den neuen zu berechnen:

```
int x = 4;
x = x + 2; // x hat jetzt den Wert 6.
x = 2 * x; // x hat jetzt den Wert 12.
```

Für den häufig genutzten Rechenschritt `i = i + 1;` gibt es sogar eine Kurzform:

```
int i = 0;
i++; // entspricht x = x + 1.
++i; // entspricht ebenfalls x = x + 1.
```

PROJEKT 9

Finger-Disko

DU BRAUCHST

- **KosmoDuino im Interaction Board**
- **Bewegungssensor**

VORBEREITUNG

Mit dem Bewegungssensor kann dein KosmoDuino auch kleinste Bewegungen registrieren. Um ein Gefühl dafür zu entwickeln, steckst du zunächst den Bewegungssensor ins Interaction Board. Lade dann den Sketch »SensorLesen« auf deinen KosmoDuino.

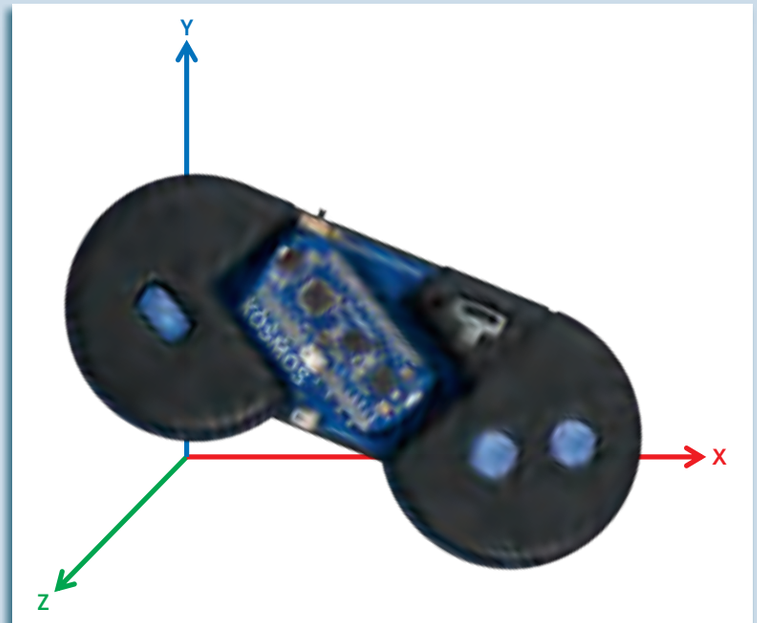
Im seriellen Monitor kannst du nun die Messwerte des Bewegungssensors beobachten. Solange du den Controller nicht bewegst, schwankt der Messwert leicht um einen Mittelwert.

Bewege den Controller nun in verschiedene Richtungen, drehe und kippe ihn. Du wirst feststellen, dass es Bewegungs- und Drehrichtungen gibt, die eine starke Veränderung des Messwertes verursachen. In andere Richtungen änderst sich der Wert kaum.

Trommel jetzt erst vorsichtig, dann etwas kräftiger mit den Fingern auf das Gehäuse des Interaction Boards. Obwohl die Erschütterungen, die deine Finger auslösen, nur sehr klein sind, werden sie vom Sensor erkannt. Wenn du den Controller auf den Tisch legst, reicht es meist sogar, nur auf den Tisch zu trommeln. Auch das wird vom Sensor bemerkt.

▲ *Finger trommeln auf Platine, NeoPixel blinkt.*

Das machen wir uns nun zu Nutze, um deinen KosmoDuino in eine Finger-Trommel-Disko-Beleuchtung zu verwandeln. Dazu benutzt du das folgende Programm:



Tatsächlich misst der Sensor lediglich die Beschleunigung des Sensors längs der sogenannten z-Achse, also der »Vor-Zurück-Achse«. Deshalb verändert sich der Wert, wenn du den Controller nach vorne oder zu dir hin bewegst. Bei Bewegungen nach oben oder unten verändert sich der Wert dagegen nicht.

DER PLAN

Bunte Lichter tanzen im Rhythmus deiner Finger. Mit diesem Projekt verwandelst du deinen KosmoDuino in eine *Finger-Disko*.

DAS PROGRAMM

Zunächst bindest du mit den `#include`-Anweisungen wieder die benötigten Bibliotheken ein, definierst `sensorPin`, um die Sensorwerte auslesen zu können, sowie `pixel`, um den NeoPixel ansteuern zu können.

Um Abweichungen vom Ruhezustand erkennen zu können, definierst du noch die Variable `ruhewert`. Darin wird später der Mittelwert in Ruhe gespeichert.

```
#include <Adafruit_NeoPixel.h>
#include <KosmoBits_Pins.h>
#include <KosmoBits_Pixel.h>

const int sensorPin = KOSMOBITS_SENSOR_PIN;
KosmoBits_Pixel pixel;

int ruhewert = 0;
```

PROJEKT 9

Um den Ruhewert zu bestimmen, soll der Mittelwert aus 20 Messwerten berechnet werden. Die Zahl der Messwerte wird in der Konstanten N gespeichert, so dass man sie leicht ändern kann. Der Mittelwert wird in der Variablen `mittelwert` gespeichert.

Die `for`-Schleife ist dir mittlerweile geläufig. Sie wird N-mal, hier also 20-mal, durchlaufen.

In jedem Durchlauf wird dem aktuellen `mittelwert` ein weiterer Messwert hinzu addiert. Vor der nächsten Messung wird 10 Millisekunden lang gewartet.

Wenn die Schleife verlassen wird, ist in der Variablen `mittelwert` die Summe der 20 Messwerte gespeichert. Um daraus den Mittelwert zu berechnen, muss die Summe noch durch die Zahl der Messwerte geteilt werden. Das geschieht in der Zeile `mittelwert = mittelwert / N;`.

Schließlich wird mit `return mittelwert;` der berechnete Mittelwert zurückgegeben.

```
int bestimmeRuhewert() {
  const int N = 20; // Zahl der Messwerte
  float mittelwert = 0;
  for (int i = 0; i < 20; ++i) {
    mittelwert = mittelwert +
    analogRead(sensorPin);
    delay(10);
  }
  mittelwert = mittelwert / N;
  return mittelwert;
}
```

Das ist einfach: Mit `pinMode(sensorPin, INPUT);` wird der `sensorPin` als Input-Pin gesetzt. Schließlich wird der Ruhewert bestimmt und in der Variablen `ruhewert` gespeichert.

```
void setup() {
  pinMode(sensorPin, INPUT);
  ruhewert = bestimmeRuhewert();
}
```

In der Hauptschleife wird zunächst ein aktueller Sensorwert gelesen und in `wert` abgespeichert. Jetzt soll die Abweichung vom Mittelwert bestimmt und in der Variablen `differenz` gespeichert werden. Die Abweichung soll immer positiv sein. Da du nicht weißt, ob der aktuelle Sensorwert größer oder kleiner als `ruhewert` ist, wird mit Hilfe der `if`-Anweisung zwischen zwei Fällen unterschieden: Ist `wert` größer als `ruhewert`, so wird `differenz` als `wert - ruhewert` berechnet, sonst als `ruhewert - wert`.

Mit `pixel.setColor()` wird nun abhängig vom `differenz`-Wert die Farbe des NeoPixels eingestellt: Je größer die Differenz, umso höher wird der Rotanteil, und umso geringer der Grünanteil.

```
void loop() {
  int wert = analogRead(sensorPin);
  int differenz;
  if (wert > ruhewert) {
    differenz = wert - ruhewert;
  } else {
    differenz = ruhewert - wert;
  }

  pixel.setColor(2 * differenz, 100 - 2 * diffe-
  renz, 0, differenz); // RGB-Wert und Helligkeit
  // einstellen.
  delay(10);
}
```

Hast du eigene Ideen, wie die Farbe des NeoPixels auf die Abweichung vom Ruhewert reagieren soll? Probiere es aus! Dir fallen bestimmt viele interessante Varianten ein.

PROJEKT 10

Piep!

Mit dem Interaction Board kannst du auch Töne erzeugen. Wie das geht, erfährst du in diesem kleinen Projekt.

DU BRAUCHST

› **KosmoDuino im Interaction Board**

DER PLAN

Der KosmoDuino piept in regelmäßigen Abständen. Keine Sorge, später lernst du auch noch, wie man eine Sirene oder gar ein Musikinstrument aus ihm macht.

DAS PROGRAMM

Zunächst bindest du die Datei `KosmoBits_Pins.h` ein, und definierst dir dann die Konstante `buzzerPin`, um den Pin anzusprechen, an den der »Buzzer«, also der Miniaturlautsprecher, angeschlossen ist.

```
#include <KosmoBits_Pins.h>

const int buzzerPin = KOSMOBITS_BUZZER_PIN;
```

In `setup()` ist diesmal nichts zu tun.

```
void setup() {
  // Nichts zu tun.
}
```

Der Aufbau der Hauptschleife `loop()` kommt dir vermutlich bekannt vor. Er entspricht dem Programm »Blink«, das du als allererstes Programm auf deinen KosmoDuino geladen hast. Anstatt aber eine LED blinken zu lassen, machst du diesmal Töne. Dazu dienen zwei Befehle:

`tone(buzzerPin, 440);` erzeugt einen Ton mit der Frequenz 440 Hz. Das ist der Kammerton A. Der Ton bleibt solange eingeschaltet, bis du ihn mit `noTone(buzzerPin);` wieder ausschaltest. Damit der Ton regelmäßig ein- und ausgeschaltet wird, hängst du an jeden dieser Befehle mit `delay(500);` eine Pause von einer halben Sekunde Länge.

```
void loop() {
  tone(buzzerPin, 440);
  delay(500);
  noTone(buzzerPin);
  delay(500);
}
```

PROJEKT 11

Zufallstöne!

Wie du deinen KosmoDuino zum Piepen bringst, hast du gerade im Projekt »Piep!« gelernt. Aber immer wieder der selbe Ton, das ist auf Dauer doch etwas langweilig. Hier lernst du, wie du dem Controller Zufallstöne entlocken kannst.

DU BRAUCHST

› **KosmoDuino im Interaction Board**

DER PLAN

Sowohl Tonhöhe als auch die Dauer des jeweiligen Tones werden vom Zufallsgenerator bestimmt.

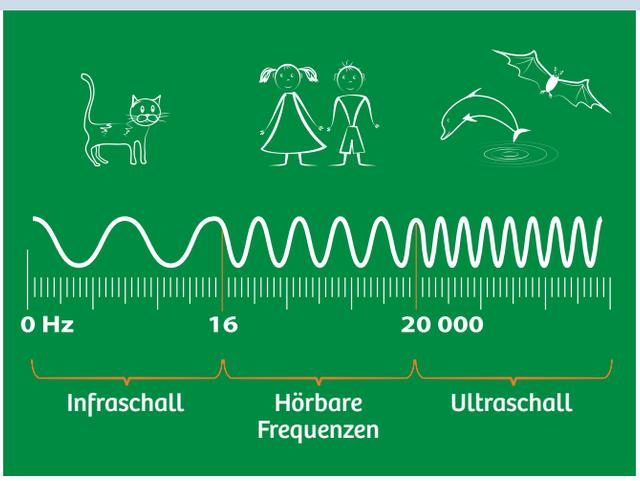
DAS PROGRAMM

Das Programm ist ziemlich simpel. In `setup()` übergibst du dem Zufallsgenerator einen Startwert, wie du es schon beim Würfel-Projekt getan hast.

In der Hauptschleife wird dann zunächst mit `int freq = random(110,1000);` die Frequenz `freq` des nächsten Tons zufällig festgelegt, danach auf gleiche Weise die Dauer.

Schließlich wird der Ton mit `tone(buzzerPin,freq);` ausgegeben. `delay(dauer);` am Ende sorgt dafür, dass der Ton für eben die Dauer `dauer` gespielt wird.

```
#include <KosmoBits_Pins.h>

const int buzzerPin = KOSMOBITS_BUZZER_PIN;

void setup() {
  randomSeed(analogRead(12));
}

void loop() {
  int freq = random(110, 1000);
  int dauer = random(10, 150);
  tone(buzzerPin, freq);
  delay(dauer);
}
```



SCHALL

Wenn die Luft, die auf unsere Trommelfelle drückt, hin und her schwingt, nehmen wir das als Ton wahr. Je schneller die Luft dabei schwingt, umso höher klingt der Ton. Physiker messen das als »Schwingungen pro Sekunde«, genannt Frequenz. Die Einheit dafür heißt Hertz, abgekürzt Hz. Ein Hertz bedeutet, dass die Luft pro Sekunde eine ganze Schwingung vollendet. Je höher die Frequenz eines Tons, umso höher klingt auch der Ton. Der Mensch kann Töne mit Frequenzen zwischen 16 und 20.000 Hertz wahrnehmen. Viele Tiere sind aber auch in der Lage Töne im Infra- und Ultraschallbereich wahrzunehmen.

PROJEKT 12

Sirene

Jetzt bringst du etwas Ordnung in das akustische Chaos: Mit diesem Projekt lässt du deinen KosmoDuino wie eine Sirene aufheulen.

DU BRAUCHST

› **KosmoDuino im Interaction Board**

DER PLAN

Bei jedem Durchlauf der Hauptschleife wird ein neuer Ton ausgegeben. Dabei wird die Tonhöhe jedesmal etwas erhöht. Ist eine bestimmte Frequenz, also Tonhöhe, erreicht, springt die Tonhöhe wieder zum Anfangswert. Das Ganze beginnt dann von vorne.

DAS PROGRAMM

Hier legst du die Frequenzen fest, innerhalb derer sich die Tonhöhen bewegen sollen. `FREQ_MIN` entspricht dem tiefsten Ton der Sirene, `FREQ_MAX` dem höchsten. Du kannst mit diesen Parametern auch experimentieren, indem du die Werte änderst und so den Klang der Sirene deinem Geschmack anpasst.

`DELAY` bestimmt, wie lange ein einzelner Ton gehalten werden soll.

Die Variable `freq` schließlich speichert die aktuelle Frequenz. Gestartet wird bei `FREQ_MIN`.

```
#include <KosmoBits_Pins.h>

const int buzzerPin = KOSMOBITS_BUZZER_PIN;
const int FREQ_MIN = 440;
const int FREQ_MAX = 660;
const int DELAY = 4;

int freq = FREQ_MIN; // Frequenz des Tones, der
// ausgegeben wird.
```

In `setup()` ist diesmal nichts zu tun.

```
void setup() {
  // Nichts zu tun.
}
```

In der Hauptschleife wird zunächst ein Ton mit der Frequenz `freq` ausgegeben (1). Nach einer Pause der Länge `DELAY` wird dann die Frequenz `freq` mit `++freq;` um den Wert 1 erhöht. In der `if`-Anweisung wird dann überprüft, ob die maximale Frequenz schon überschritten wurde. Wenn ja, wird die Frequenz `freq` wieder auf den Startwert `FREQ_MIN` zurückgesetzt.

```
void loop() {
  tone(buzzerPin, freq); // (1)
  delay(DELAY);
  ++freq; // Kurz für: freq = freq + 1;
  if (freq > FREQ_MAX) {
    freq = FREQ_MIN;
  }
}
```

Im nächsten Projekt wirst du eine ganze Tonleiter erklingen lassen. Aber vorher musst du noch ein wichtiges Hilfsmittel der Programmierung kennenlernen: Arrays.

WISSEN KOMPAKT

Arrays

Wenn du viele Werte vom gleichen Typ speichern möchtest, zum Beispiel die letzten 100 Messwerte, wird es mit einfachen Variablen schnell unübersichtlich und aufwendig zu tippen. Du müsstest schließlich für jeden einzelnen Wert eine eigene Variable anlegen, also zum Beispiel:

```
int wert1;
int wert2;
int wert3;
...usw...
int wert 100;
```

Viel einfacher ist es, stattdessen ein sogenanntes *Array* (gesprochen »Ärräj« mit weichem, englischem »r«) zu benutzen.

In einem Array kannst du nämlich gleich mehrere Werte speichern. Damit du auf die verschiedenen Werte zugreifen kannst, werden sie im Array durchnummeriert. Du kannst dir ein Array also als eine Art Tabelle vorstellen, in der die einzelnen Werte in nummerierte Spalten eingetragen werden. Die Nummerierung der Spalten beginnt allerdings nicht mit 1, sondern mit 0:

INDEX	0	1	2	3	...	n
WERTE	Wert 0	Wert 1	Wert 2	Wert 3	...	Wert n

*Ein Array wird im Deutschen manchmal auch **Feld** oder **Datenfeld** genannt. Allerdings ist die englische Bezeichnung auch im Deutschen sehr viel gängiger.*

Ein Array legst du allgemein folgendermaßen an:

```
typ arrayName[länge];
```

Der Typ legt fest, welche Art von Daten in dem Array gespeichert werden können (int, float, usw.). Die Anzahl der Werte, die in einem Array gespeichert werden können, wird als *Länge* bezeichnet.

Um also zum Beispiel ein Array anzulegen, das 10 Integerwerte aufnehmen kann, gehst du so vor:

```
int meinArray[10]; // Ein Array, das 10
// int Werte aufnehmen kann
```

Du kannst einem Array auch gleich bei der Definition Werte zuweisen, indem du die einzelnen Werte in geschweiften Klammern angibst:

```
int meinArray[4] = {0, 4, 2, 3};
```

In dem Fall brauchst du nicht unbedingt anzugeben, wie viele Werte es sind. Du kannst also auch so schreiben:

```
int meinArray[] = {0, 4, 2, 3};
```

WISSEN KOMPAKT

Häufig möchtest du, dass alle einzelnen Werte auf Null gesetzt werden. Das kannst du kurz so schreiben:

```
int meinArray[10] = {}; // Array der
// Länge 10. Alle Werte werden auf 0
// gesetzt.
```

Um auf die einzelnen Werte eines Arrays zuzugreifen, gibst du in eckigen Klammern den Index des Wertes an, auf den du zugreifen möchtest. Wenn du dir das Array also als Tabelle vorstellst, entspricht der Index der Nummer der Spalte.

```
int meinArray[100] = {}; // Array der
// Länge 100; alle Werte 0.
meinArray[2] = 42; // Der Wert an Index-
// Position 2 ist jetzt 42.

int x = meinArray[2]; // x erhält den in
// meinArray an Index-Position 2
// gespeicherten Wert, hier also 42.
```

Häufig möchtest du aus den einzelnen Werten einen neuen berechnen, zum Beispiel die Summe aller Einzelwerte. Da du die einzelnen Werte über ihren Index ansprechen kannst, geht das nun mit einer `for`-Schleife sehr einfach:

```
// werte ist ein Array der Länge 100.

int summe = 0;
for (int index = 0; index < 100;
++index) {
  summe = summe + werte[index];
}
```

Beim ersten Durchlauf der Schleife wird der Wert `werte[0]` zu `summe` hinzu addiert. Danach wird `index` um 1 erhöht, und der nächste Wert addiert. Das wird solange wiederholt, bis der letzte Wert, also `werte[99]`, addiert wurde.

Im nächsten Projekt wirst du ein Array benutzen, um eine Tonleiter abzuspielen.

ACHTUNG!

Da der Index für den ersten Wert nicht 1 ist, sondern 0, ist der höchstmögliche Index für ein Array der Länge N nicht N, sondern N - 1. Bei einem Array der Länge 100, wäre also der höchstmögliche Index 99. Benutzt du einen höheren Index, kommt es zu einem Fehler im Programmablauf: Was dann passiert lässt sich nicht vorhersagen. Achte also darauf, dass du nur gültige Werte für den Index benutzt.

PROJEKT 13

Tonleiter

DU BRAUCHST

› **KosmoDuino im Interaction Board**

DER PLAN

Dein KosmoDuino spielt eine Tonleiter.

DAS PROGRAMM

Zunächst bindest die Datei `KosmoBits_Pins.h` ein und definierst dir die Konstanten `buzzerPin` und `freqGrundton`. Schließlich legst du noch das Array `tonleiter[]` an, in dem die Frequenzverhältnisse der Töne einer Dur-Tonleiter abgelegt werden. Um die Frequenz eines Tones zu ermitteln, musst du den entsprechenden Wert mit der Frequenz des Grundtones multiplizieren.

```
#include <KosmoBits_Pins.h>

const int buzzerPin = KOSMOBITS_BUZZER_PIN;
const int freqGrundton = 220; // Das ist der
// Ton A (eine Oktave tiefer als der Kammer-
// ton A aus Projekt 10).

float tonleiter[] = {1.f, 9.f/8, 5.f/4, 4.f/3,
3.f/2, 5.f/3, 15.f/8, 2.f};
```

Diesmal passiert alles bereits in `setup()`. In der `for`-Schleife wird die Variable `i` von 0 bis 7 hochgezählt. Für jeden Wert von `i` wird dann mit `tone(buzzerPin, tonleiter[i] * freqGrundton);` der `i`-te Ton der Tonleiter ausgegeben. Wie oben bereits beschrieben, wird zur Ermittlung der richtigen Frequenz die Frequenz des Grundtons `freqGrundton` mit dem entsprechenden Frequenzverhältnis `tonleiter[i]` multipliziert.

Nach einer Pause wird die `for`-Schleife erneut durchlaufen, bis der höchste Ton der Tonleiter erreicht wurde. Danach ist Stille: `noTone(buzzerPin);`

```
void setup() {
  for (int i = 0; i < 8; ++i) {
    tone(buzzerPin, tonleiter[i] * freqGrundton);
    delay(500);
  }
  noTone(buzzerPin);
}
```

In der Hauptschleife wird diesmal nichts gemacht. Die Tonleiter wird also nur ein einziges Mal abgespielt. Um die Tonleiter nochmal abzuspielen, kannst du die Reset-Taste auf dem KosmoDuino drücken. Dadurch wird der Code zurückgesetzt und von Neuem ausgeführt.

```
void loop() {
  // Hier wird nichts gemacht.
}
```

PROJEKT 14

Sensor-Orgel

Nachdem du nun gelernt hast, wie man Tonleitern spielt, kannst du in diesem Projekt ein richtiges Musikinstrument programmieren.

DU BRAUCHST

› **KosmoDuino im Interaction Board**
› **Bewegungssensor**

DER PLAN

Per Tastendruck erzeugst du Töne. Die Tonhöhe kannst du verändern, indem du das Interaction Board bewegst. Mit ein wenig Übung kannst du so richtige Melodien spielen.

DAS PROGRAMM

Zunächst definierst du ein paar Konstanten für Sensor-, Buzzer- und Taster-Pin. Danach legst du wie im vorigen Projekt ein Array für die Tonleiter, sowie eine Konstante für den Grundton, an.

Mit den Konstanten `messwertMin` und `messwertMax` stellst du ein, wie weit der Controller gekippt werden muss, um den höchsten bzw. den tiefsten Ton zu spielen. Du kannst diese Werte später für dich anpassen. Die Konstante `skalierung` benutzt du später, um aus einem Messwert die Tonhöhe zu bestimmen. Du teilst damit quasi den gesamten Messbereich in 9 Bereiche ein.

Der Bewegungssensor ist recht empfindlich. Deshalb schwanken die Messwerte ein wenig. Damit die Tonhöhe dadurch nicht zu wackelig wird, nimmst du nicht direkt den letzten Messwert, um daraus die Tonhöhe zu berechnen, sondern bestimmst den Mittelwert von 10 Messungen. Die letzten 10 Messwerte werden dazu in dem Array `werte[]` gespeichert. Durch `int werte[N] = {};` werden alle Werte darin zunächst auf 0 gesetzt.

Mit der Variablen `index` wird das Array durchlaufen. Gestartet wird wie üblich bei 0.

```
#include <KosmoBits_Pins.h>

const int sensorPin = KOSMOBITS_SENSOR_PIN;
const int buzzerPin = KOSMOBITS_BUZZER_PIN;
const int tasterPin = KOSMOBITS_TASTE_2_PIN;

const float tonleiter[] = {1.f, 9.f/8, 5.f/4,
4.f/3, 3.f/2, 5.f/3, 15.f/8, 2.f};
const int freqGrundton = 220; // Das ist der
// Ton A.

const int messwertMin = 300;
const int messwertMax = 400;
const int skalierung = (messwertMax - messwert
Min) / 9;

const int N = 10; // Anzahl der Werte, die
// gemittelt werden.
int werte[N] = {}; // Hier werden die letzten
// 10 Werte gespeichert.
int index = 0;
```

Hier werden nur die jeweiligen Pin-Modi eingestellt.

```
void setup() {
  pinMode(sensorPin, INPUT);
  pinMode(tasterPin, INPUT);
}
```

Zunächst liest du einen neuen Messwert und speicherst ihn im Array `werte` an Index-Position `index`. Damit der nächste Messwert an der darauffolgenden Position gespeichert wird, wird mit `++index;` die Variable `index` um 1 erhöht. Wird dabei der Wert `N` erreicht, stellt die

```
void loop() {
  // Lese einen neuen Wert vom Sensor und
  // speichere ihn in werte[].
  werte[index] = analogRead(sensorPin);
```

PROJEKT 14

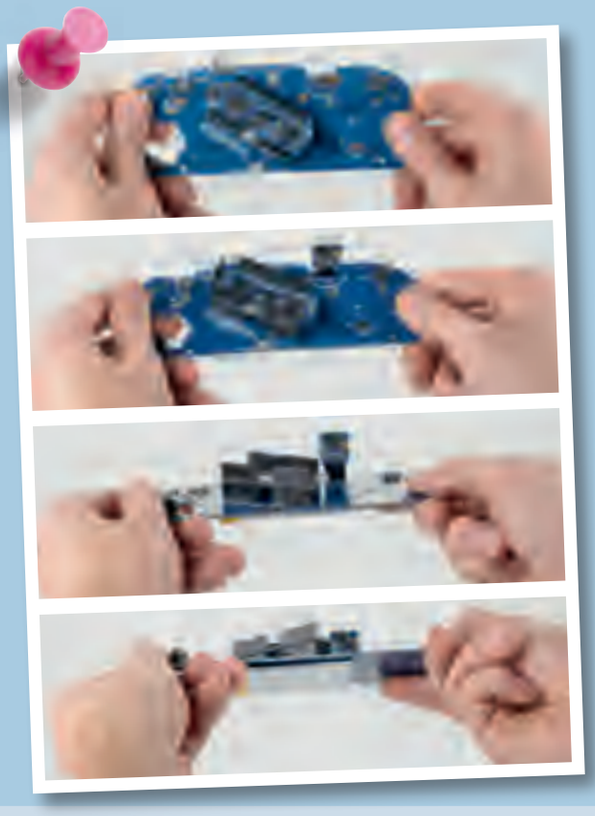
`if`-Anweisung `index` auf 0 zurück. Das Array wird wieder von vorne befüllt.

Danach geht es daran, den Mittelwert der letzten 10 Messwerte zu bestimmen. Dazu wird die Variable `mittelwert` zunächst auf 0 gesetzt. In der `for`-Schleife werden dann alle im Array `werte[]` gespeicherten Messwerte zu `mittelwert` hinzu addiert. Um daraus dann den Mittelwert zu bestimmen, muss das Ergebnis noch durch die Zahl der Werte geteilt werden:
`mittelwert = mittelwert / N;`.

Mit `int tonIndex = (mittelwert - messwertMin) / skalierung;` ermittelst du den »Tonindex«: Der wievielte Ton der Tonleiter soll gespielt werden?

Es kann sein, dass `tonIndex` einen ungültigen Wert hat, also kleiner als 0 oder größer als 7 ist. Deshalb korrigierst du das im Zweifelsfall mit der `if-else`-Anweisung.

Schließlich wird in der letzten `if`-Abfrage getestet, ob der Taster gedrückt ist. Wenn ja, wird der entsprechende Ton ausgegeben. Wenn nicht, wird kein Ton ausgegeben.



```
  // Erhöhe den Index um 1, damit der nächste
  // Wert an der nächsten Stelle
  // im Feld werte[] gespeichert werden kann.
  ++index;

  // Wenn index == N ist, ist das Feld voll
  // beschrieben.
  // index wird dann auf 0 gesetzt, damit im
  // nächsten Durchlauf
  // der älteste Wert überschrieben wird.
  if (index == N) {
      index = 0;
  }

  // Jetzt wird der Mittelwert der letzten N
  // Messwerte berechnet.
  int mittelwert  = 0; // Starte mit dem Wert 0

  // Summiere die letzten 10 gemessenen Werte.
  for (int i = 0; i < N; ++i) {
    mittelwert = mittelwert + werte[i];
  }
  // Teile die Summe durch die Zahl der Werte.
  mittelwert = mittelwert / N;

  int tonIndex = (mittelwert - messwertMin) /
  skalierung;

  // tonIndex darf nicht kleiner 0 und nicht
  // größer 7 sein:
  if (tonIndex < 0) {
    tonIndex = 0;
  } else if (tonIndex > 7) {
    tonIndex = 7;
  }

  // Ton ausgeben
  if (digitalRead(tasterPin) == LOW) {
    tone(buzzerPin, tonleiter[tonIndex] * freq-
Grundton);
    delay(10);
  } else {
    noTone(buzzerPin);
    delay(1);
  }
}
```

Zugegeben: Das war schon etwas umfangreicher.
Das Ergebnis lohnt sich aber!

PROJEKT 15

Der serielle Plotter

DU BRAUCHST

- **KosmoDuino im Interaction Board**
- **Schallsensor**

DER PLAN

Bei dem Schallsensor handelt es sich im Grunde um ein kleines Mikrofon, das du über den Sensor-Pin auslesen kannst. Um zu verstehen, wie er sich verhält, benutzt du am besten ein weiteres tolles Werkzeug der Arduino-Umgebung, nämlich den *seriellen Plotter*.

DAS PROGRAMM

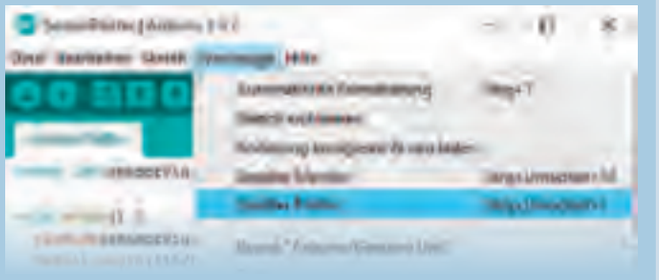
Stecke den Schallsensor in das Interaction Board und lade dann das folgende kleine Programm auf deinen KosmoDuino:

Besonders viel macht der Code nicht. Bei jedem Durchlauf der Hauptschleife wird lediglich ein neuer Wert vom Sensor-Pin gelesen und mit `Serial.println()` ausgegeben.

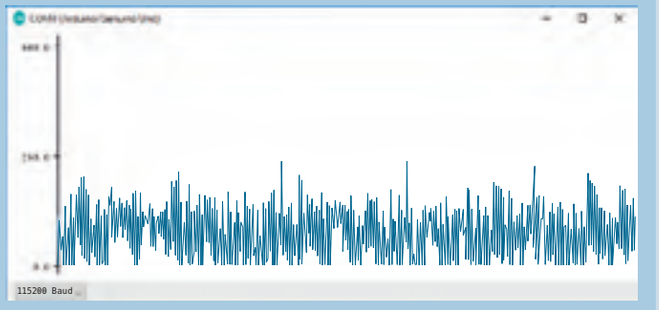
```
#include <KosmoBits_Pins.h>

const int sensorPin = KOSMOBITS_SENSOR_PIN;

void setup() {
  pinMode(sensorPin, INPUT);
  Serial.begin(115200);
}

void loop() {
  Serial.println(analogRead(sensorPin));
}
```

Starte jetzt aus dem Menü »Werkzeuge« den *Seriellen Plotter*.

Es öffnet sich ein neues Fenster, in dem es ganz schön wild zu geht.



Versuche jetzt einmal verschiedene Geräusche zu machen, zu pfeifen oder zu singen. Du wirst sehen, wie sich das Bild im seriellen Plotter verändert.



An der Skala auf der linken Seite kannst du leicht ablesen, wie groß die Werte sind, die der Sensor liefert. Im hier abgebildeten Beispiel siehst du, dass 0 der niedrigste Wert ist. Das ist auch klar, etwas kleineres kann `analogRead()` nicht liefern. Der größte gemessene Wert ist ungefähr 250. Wenn du aber in die Hände klatschst, können die gemessenen Werte auch mal auf 800 ansteigen.

Das kannst du im folgenden Projekt prima ausnutzen!

PROJEKT 16

Der Klatschschalter

Wäre es nicht cool, wenn dein KosmoDuino auf Klatschen reagieren würde? Mit Hilfe des Schallsensors kannst du das leicht programmieren.

DU BRAUCHST

- **KosmoDuino im Interaction Board**
- **Schallsensor**

DER PLAN

Wenn du einmal klatschst, soll der NeoPixel auf dem Interaction Board eingeschaltet werden. Wenn du dann noch einmal klatschst, geht er wieder aus.

DAS PROGRAMM

Es werden die üblichen Dateien eingebunden sowie die Konstante `sensorPin` für den Sensor-Pin definiert.
Die Konstante `schwelle` regelt die Empfindlichkeit deines Klatschschalters. Wird der hier eingestellte Wert vom Messwert überschritten, so schaltet der Schalter um. Entweder von »an« auf »aus«, oder eben umgekehrt.
In der Variablen `an` speicherst du den aktuellen Zustand des NeoPixels ab. Ist er an, so wird ihr der Wert `true` (wahr) zugewiesen, andernfalls der Wert `false` (falsch).
Zu Beginn des Programms ist der NeoPixel aus: `bool an = false;`.

```
#include <KosmoBits_Pins.h>
#include <Adafruit_NeoPixel.h>
#include <KosmoBits_Pixel.h>

const int sensorPin = KOSMOBITS_SENSOR_PIN;
const int schwelle = 500;

KosmoBits_Pixel pixel;

bool an = false;
```

In `setup()` wird der Pin-Modus des Sensor-Pins eingestellt und der NeoPixel ausgeschaltet.

```
void setup() {
  pinMode(sensorPin, INPUT);
  pixel.setColor(0, 0, 0, 0);
}
```

In der Hauptschleife wird fortwährend der Schallsensor ausgelesen. Ist der gemessene Wert größer als der in `schwelle` eingestellte Wert, so wird die Funktion `umschalten()` aufgerufen und anschließend kurz gewartet.

```
void loop() {
  int sensorWert = analogRead(sensorPin);
  Serial.println(sensorWert);
  if (sensorWert > schwelle) {
    umschalten();
    delay(200);
  }
}
```

Hier schreibst du dir deine eigene Funktion `void umschalten()`. Es wird dort abgefragt, ob der NeoPixel derzeit an ist. Wenn ja, wird der NeoPixel ausgeschaltet und `an` der Wert `false` zugewiesen.
Andernfalls (»else«) wird der NeoPixel eingeschaltet und `an` der Wert `true` (wahr) zugewiesen.

Kurz: `umschalten()` macht genau das, was der Name vermuten lässt. Es wird zwischen "ein" und "aus" hin und her geschaltet.

```
void umschalten() {
  if (an) {
    pixel.setColor(0, 0, 0, 0);
    an = false;
  } else {
    pixel.setColor(255, 0, 0, 30);
    an = true;
  }
}
```

PROJEKT 17

Der Schubladenwächter

Hattest du schon immer den Verdacht, dass dein Bruder oder deine Schwester heimlich deine Schublade durchwühlt, wenn du nicht zu Hause bist? Mit dem Schubladenwächter Projekt kannst du ihn überführen! Das Gute daran: Der Täter merkt nicht einmal, dass die Schublade überwacht wird: Es gibt keinen Alarm, nichts weist den Täter darauf hin, dass seine Tat bemerkt wurde. Erst, wenn du den KosmoDuino befragst, zeigt er dir an, ob sich jemand an deinen Geheimnissen zu schaffen gemacht hat!

DU BRAUCHST

- **KosmoDuino im Interaction Board**
- **Lichtsensor**

DER PLAN

In einer geschlossenen Schublade ist es dunkel, in einer offenen nicht. Mit dem Lichtsensor kannst du also feststellen, ob deine Schublade geöffnet wurde oder nicht.

Du wirst deinen KosmoDuino so programmieren, dass er folgendes für dich macht:

1. Nach dem Einschalten wartet er darauf, durch einen Druck auf Taste 1 »scharf« gestellt zu werden.
2. Ist der Schubladenwächter scharf geschaltet, wartet er, bis die Schublade geschlossen wurde. Erst dann beginnt er mit der Überwachung der Schublade.
3. Wird die Schublade dann wieder geöffnet, gibt es zwei Möglichkeiten, wie es weitergeht:

   a) Wird die Schublade wieder geschlossen, so wird dies als »Schublade wurde geöffnet« gezählt.

   b) Drückst du auf Taste 2, so wird dir durch den NeoPixel angezeigt, wie oft die Schublade geöffnet wurde.

VORBEREITUNG: SENSORWERTE MESSEN

Um zu bemerken, ob die Schublade geöffnet wurde, benutzt du den Lichtsensor. Er misst die Helligkeit des Lichts, das auf ihn fällt. Solange sich dein KosmoDuino in der dunklen Schublade befindet, wird er einen niedrigen Wert messen. Wird die Schublade aber geöffnet, erhöht sich der gemessene Wert. Leider ist es nicht in jeder geschlossenen

Schublade immer vollständig dunkel. Ab welchem Helligkeitswert ist die Schublade aber geöffnet? Das probierst du nun aus! Stecke dazu als Erstes den Lichtsensor in das Interaction Board. Jetzt musst du herausfinden, wie hell es im Hellen ist. Das machst du, indem du den Sensor ausliest und den Messwert über den seriellen Monitor ausgibst. Das machst du wie immer mit dem Sketch **SensorLesen**.

Notiere dir, welche Messwerte der Sketch ausgibt, wenn du den Lichtsensor

- *dem Tageslicht aussetzt,*
- *der Dämmerung aussetzt,*
- *mit der Hand abdeckst,*
- *mit einer Taschenlampe bestrahlst.*

LICHT	MESSWERT
Tageslicht	
Dämmerung	
abgedunkelt	
Taschenlampe	

Du wirst diese Werte später benötigen, wenn du dir überlegst, wann die Schublade als geschlossen gewertet werden soll.

PROJEKT 17

DAS GRUNDGERÜST

Wir benutzen in diesem Projekt den NeoPixel und die beiden Tasten. Deshalb werden hier die benötigten Bibliotheken eingebunden.

```
#include <Adafruit_NeoPixel.h>
#include <KosmoBits_Pins.h>
#include <KosmoBits_Pixel.h>
```

Wir definieren drei Konstanten, um den Sensor und die Tasten auszulesen, sowie die drei Variablen `taste1`, `taste2` und `sensorWert`, in dem die aktuellen Messwerte gespeichert werden.

```
const int sensorPin = KOSMOBITS_SENSOR_PIN;
const int taste1Pin = KOSMOBITS_TASTE_1_PIN;
const int taste2Pin = KOSMOBITS_TASTE_2_PIN;

int taste1 = HIGH;
int taste2 = HIGH;
int sensorWert = 0;
```

Damit der Programm-Code übersichtlich bleibt, hilft es, sich das Programm als eine Maschine vorzustellen, die in unterschiedlichen Betriebsarten (Modi) betrieben werden kann. Je nach Modus wird in der Hauptschleife etwas anderes gemacht.

Dazu definieren wir mit dem Wort `enum Modus` einen sogenannten Aufzählungstypen: Eine Variable des Typs `Modus` kann nur die Werte annehmen, die in den geschweiften Klammern stehen, also `LEERLAUF`, `COUNTDOWN` usw.
Anschließend definieren wir die Variable `modus` in der der aktuelle Betriebsmodus gespeichert wird. Wir beginnen im Modus `LEERLAUF`.

```
enum Modus {LEERLAUF, COUNTDOWN, WARTE_BIS_
DUNKEL, GESCHLOSSEN, OFFEN};

Modus modus = LEERLAUF;
```

Die Konstante `DUNKEL_WERT` legt fest, ab welchem Messwert des Lichtsensors die Schublade als geschlossen gewertet wird. Manche Schubladen lassen immer noch ein wenig Licht hinein. **Passe den Wert also so an, dass er für deine Schublade passend ist**. Die Werte, die du zur Vorbereitung gemessen hast, helfen dir dabei!

In der Variablen `zaehler` wird gespeichert, wie oft die Schublade geöffnet wurde. Schließlich verschaffen wir uns mit `pixel` noch Zugriff auf den NeoPixel.

```
const int DUNKEL_WERT = 5;
unsigned long zaehler = 0;
KosmoBits_Pixel pixel;
```

Wenn die Überwachung beendet wurde, möchten wir wieder in den Ausgangszustand zurückkehren. Also so, als ob wir den KosmoDuino gerade erst eingeschaltet hätten. Dafür führen wir die Funktion `reset()` ein. Der NeoPixel wird ausgeschaltet, `modus` auf `LEERLAUF` und `zaehler` zurück auf `0` gesetzt. Zur Kontrolle wird über die serielle Schnittstelle »Reset!« ausgegeben.

```
void reset() {
  pixel.setColor(0, 0, 0, 0); // Pixel aus
  modus = LEERLAUF;
  zaehler = 0;
  Serial.println("Reset!");
}
```

In `setup()` legen wir, wie üblich, die Betriebsarten der benutzten Pins fest und initialisieren die Ausgabe über die serielle Schnittstelle. Danach wird `reset()` aufgerufen, um den Schubladenwächter in den Ausgangszustand zu versetzen.

```
void setup() {
  pinMode(sensorPin, INPUT);
  pinMode(taste1Pin, INPUT);
  pinMode(taste2Pin, INPUT);
  Serial.begin(115200);
  reset();
}
```

In der Hauptschleife werden je nach Betriebsart verschiedene loop-Funktionen aufgerufen. Dazu dient die `switch(modus)`-Anweisung. Damit wird eine Fallunterscheidung eingeleitet. Für jeden Fall (*engl. case*), also möglichen Wert von `modus`, wird eine andere Funktion aufgerufen:

- *Hat* `modus` *den Wert* `LEERLAUF`*, wird* `loopLeerlauf()` *aufgerufen.*
- *Hat* `modus` *den Wert* `COUNTDOWN`*, wird* `loopCountdown()` *aufgerufen, usw.*

Wichtig ist die `break;`-Anweisung (*engl. abbrechen*), um die Behandlung eines Falles abzuschließen und die Fallunterscheidung zu beenden. Ohne die `break;` Anweisung würde der Code des nächsten Falles auch ausgeführt werden.

```
void loop() {
  switch(modus) {
  case LEERLAUF:
    loopLeerlauf();
    break;
  case COUNTDOWN:
    loopCountdown();
    break;
  case WARTE_BIS_DUNKEL:
    loopWarteBisDunkel();
    break;
  case GESCHLOSSEN:
    loopGeschlossen();
    break;
  case OFFEN:
    loopOffen();
    break;
  }
}
```

Damit haben wir das Grundgerüst unseres Programms fertig. Jetzt müssen wir uns nur noch überlegen, was genau der Schubladenwächter in den unterschiedlichen Betriebsarten machen soll.

◀ *Board mit Lichtsensor*

IM LEERLAUF

Im Leerlauf wartet der Schubladenwächter darauf, dass Taste 1 gedrückt wird, um die Überwachung zu starten. Dafür verantwortlich ist die Funktion `loopLeerlauf()`, die im Modus `LEERLAUF` aus der Hauptschleife aufgerufen wird.

Den Code verstehst du mittlerweile wahrscheinlich schon ohne Probleme. In der Variablen `gedrueckt` wird gespeichert, ob die Taste gedrückt wurde. Wird Taste 1 gedrückt, wird die `while()`-Schleife durchlaufen: Der NeoPixel leuchet grün, und die Variable `gedrueckt` wird auf den Wert `true` gesetzt. Wenn Taste 1 losgelassen wird, wird die `while()`-Schleife verlassen. Am Ende wird mit der `if()`-Anweisung getestet, ob die Taste gedrückt worden ist. Falls ja, wird der Modus auf `COUNTDOWN` gesetzt. Andernfalls wird `loopLeerlauf()` verlassen und wieder die Hauptschleife aufgerufen. Die ruft dann erneut `loopLeerlauf()` auf, weil sich der Modus ja nicht geändert hat.

```
void loopLeerlauf() {
  Serial.println("Leerlauf");
  bool gedrueckt = false;
  while(digitalRead(taste1Pin) == LOW) {
    // Taste 1 wurde gedrückt.
    // Warte, bis sie wieder losgelassen wird.
    pixel.setColor(0, 255, 0, 25);
    gedrueckt = true;
    delay(50);
  }
  if (gedrueckt) {
    modus = COUNTDOWN;
  }
}
```

COUNTDOWN

Auch hier dürfte dir das meiste bekannt vorkommen. Zunächst wird in der Konstanten `COUNTDOWN_ZAHL` festgelegt, wie viele Sekunden der Countdown dauern soll: 5 Sekunden. Der NeoPixel wird ausgeschaltet und nach 500 Millisekunden der eigentliche Countdown in der `for()`-schleife gestartet: 5-mal »An - Warten - Aus«. Danach wird der NeoPixel eine Sekunde (1000 Millisekunden) lang auf »rot« gesetzt. Schließlich wird der Modus auf `WARTE_BIS_DUNKEL` gesetzt und `loopCountdown()` verlassen.

```
void loopCountdown() {
  const int COUNTDOWN_ZAHL = 5;
  Serial.println("Countdown");
  pixel.setColor(0, 0, 0, 0); // Pixel aus
  delay(500);
  // Der eigentliche Countdown: Wie Blink.
  for (int i = 0; i < COUNTDOWN_ZAHL; ++i) {
    pixel.setColor(0, 255, 0, 25); // Grün
    delay(500);
    pixel.setColor(0, 0, 0, 0); // Pixel aus
    delay(500);
  }
  pixel.setColor(255, 0, 0, 25); // Rot
  delay(1000);
  pixel.setColor(0, 0, 0, 0); // Pixel aus
  modus = WARTE_BIS_DUNKEL;
}
```

WARTEN BIS ES DUNKEL IST

Das ist einfach: Es wird so lange die `while`-Schleife durchlaufen, bis es dunkel ist. Gemessen wird alle 10 Millisekunden. Wenn die `while()`-Schleife verlassen wurde, wird der Modus auf `GESCHLOSSEN` gesetzt und `loopBisDunkel()` verlassen.

```
void loopBisDunkel() {
  Serial.println("Warte bis dunkel");
  while (analogRead(sensorPin) > DUNKEL_WERT) {
    delay(10);
  }
  modus = GESCHLOSSEN;
  delay(1000);
}
```

ÜBERWACHEN (GESCHLOSSEN)

Die eigentliche Überwachung der Schublade geschieht im Modus `GESCHLOSSEN`, also in `loopGeschlossen()`. Zunächst wird die aktuelle Helligkeit gemessen. Ist es dunkel, wird eine Sekunde (1000 Millisekunden) lang gewartet. `loopGeschlossen()` wird dann aus der Hauptschleife erneut aufgerufen.

Ist es hell, so wird nach 0,5 Sekunden noch einmal gemessen. Ist es immer noch hell, werten wir das, als »Schublade ist offen«. Der Modus wird auf `OFFEN` gesetzt: Im nächsten Durchlauf der Hauptschleife wird dann `loopOffen()` aufgerufen.

```
void loopGeschlossen() {
  Serial.println("Im Dunkeln");
  sensorWert = analogRead(sensorPin);
  if (sensorWert < DUNKEL_WERT) {  // Es ist
  // noch dunkel.
    delay(1000); // Warte 1 Sekunde, loop() ruft
    // diese Funktion dann wieder auf.
  } else {
    // Es ist hell! Kontrolliere nach
    // 0,5 Sekunden noch einmal.
    delay(500);
    if (analogRead(sensorPin) >= DUNKEL_WERT) {
      // Es ist hell:Schublade geöffnet!
      modus = OFFEN;
    }
  }
}
```

EINDRINGLINGE (OFFEN)?

Ist die Schublade geöffnet worden, muss der Schubladenwächter noch herausfinden, ob es sich um einen Eindringling handelt. Vielleicht hast du die Schublade ja selbst geöffnet. Das erledigt `loopOffen()`. Diese Funktion wird aus der Hauptschleife immer dann aufgerufen, wenn `modus` den Wert `OFFEN` hat. Wir kontrollieren zunächst Taste 2: Wird sie gedrückt, dann wird das Ergebnis der Überwachung ausgegeben, indem die Funktion `ausgabe()` aufgerufen wird.

Wird Taste 2 nicht gedrückt, so wird im `else if`-Teil die Helligkeit kontrolliert: Ist es jetzt dunkel, ist die Schublade wieder geschlossen worden. `zaehler` wird dann mit `++zaehler` um den Wert 1 erhöht.

Ist weder Taste 2 gedrückt, noch die Schublade wieder geschlossen worden, wird die Funktion verlassen und von der Hauptschleife erneut aufgerufen.

```
void loopOffen() {
  Serial.println("geöffnet");
  delay(50);
  taste2 = digitalRead(taste2Pin);
  sensorWert = analogRead(sensorPin);

  if (taste2 == LOW) {
    ausgabe();
  } else if (sensorWert < DUNKEL_WERT) {
    // Schublade wieder zu.
    ++zaehler;
    modus = GESCHLOSSEN;
  }
}
```

AUSGABE

Um die Ausgabe des Überwachungsergebnisses kümmert sich die Funktion `ausgabe()`. Zunächst wird das genaue Ergebnis über die serielle Schnittstelle ausgegeben. Für die schnelle Kontrolle, ohne angeschlossenen Computer, wird das Ergebnis aber auch über den NeoPixel ausgegeben: Ist die Schublade geöffnet und wieder geschlossen worden (also `zaehler > 0`), so leuchtet der NeoPixel für 2 Sekunden lang rot. Andernfalls leuchtet er grün. Danach wird der Schubladenwächter mit `reset()` wieder in den Ausgangszustand zurückgesetzt.

```
void ausgabe() {
  Serial.print("Die Schublade wurde ");
  Serial.print(zaehler);
  Serial.print("-mal geöffnet!\n");
  if (zaehler > 0) {
    pixel.setColor(255, 0, 0, 25);
  } else {
    pixel.setColor(0, 255, 0, 25);
  }
  delay(2000);
  reset();
}
```

PROJEKT 18

ACHTUNG! Nicht geeignet für Kinder unter 10 Jahren. Falls ein inkorrekter Kurzschluss durch unterschiedliche Polaritäten verursacht, oder der Kondensator unter falschen Bedingungen verwendet wird, können heiße Oberflächen durch Bauteile auf der Platine entstehen.

In den Kühlschrank geschaut

Wie kann man eigentlich feststellen, ob das Licht im Kühlschrank wirklich aus geht, wenn du die Kühlschranktür schließt? Nachsehen kannst du ja nicht ohne weiteres. Schließlich müsstest du dazu die Tür wieder öffnen und dann ginge das Licht wieder an.
Mit deinem KosmoDuino allerdings, ist das kein Problem!

DU BRAUCHST

- **KosmoDuino im Interaction Board**
- **Lichtsensor**
- **4 Jumperkabel männlich-weiblich**

VORBEREITUNG

Löse von dem Strang mit den männlich-weiblichen Jumperkabeln einen Strang mit 4 Kabeln ab. Die Steckerenden steckst du in die Buchsen, in die du sonst die KosmoBits-Sensor-Module hineinsteckst.

In die Buchsenenden der Kabel steckst du dann den Lichtsensor. Achte unbedingt darauf, dass jeder Pin des Lichtsensor über die Jumperkabel mit genau der Buchse verbunden ist, in der er normalerweise stecken würde. Die Kabel dürfen sich also nicht überkreuzen.

DER PLAN

Mit dem Lichtsensor misst du die Helligkeit im Kühlschrank. Den NeoPixel auf dem Interaction Board lässt du entsprechend der gemessenen Helligkeit aufleuchten. Aber wie kannst du dir außerhalb des Kühlschranks anzeigen lassen, wie hell es in seinem Inneren ist? Ganz einfach: Du benutzt »Verlängerungskabel«, um den Sensor anzuschließen! Den Sensor hältst du dann mit Hilfe der Kabel in den Kühlschrank, während der KosmoDuino mit dem Interaction Board draußen bleibt.

Die Kabel sind so dünn, dass du die Kühlschranktür trotzdem schließen kannst!

PROJEKT 18

DAS PROGRAMM

Das Programm verstehst du bestimmt schon ganz alleine. Trotzdem ganz kurz: In `setup()` stellst du den Pin-Modus des Sensor-Pins ein und schaltest den NeoPixel aus. In der Hauptschleife misst du über den Licht-Sensor die Helligkeit und speicherst den Messwert in der Variablen `helligkeit`. Da der Helligkeitswert bei `pixel.setColor()` nicht größer als 255 sein darf, begrenzt du den Wert von `helligkeit` mit Hilfe der `if`-Anweisung auf 255.

Jetzt brauchst du nur noch den Sensor in den Kühlschrank zu halten und die Tür zu schließen. Und? Geht das Licht aus?

```
#include <Adafruit_NeoPixel.h>
#include <KosmoBits_Pixel.h>
#include <KosmoBits_Pins.h>

const int sensorPin = KOSMOBITS_SENSOR_PIN;
KosmoBits_Pixel pixel;

void setup() {
  pinMode(sensorPin, INPUT);
  pixel.setColor(255, 255, 255, 0);
}

void loop() {
  int helligkeit = analogRead(sensorPin);
  if (helligkeit > 255) {
    helligkeit = 255;
  }
  pixel.setColor(255, 255, 255, helligkeit);
  delay(10);
}
```

▲ *Jumperkabel in Kühlschrank*

PROJEKT 19

ACHTUNG! Nicht geeignet für Kinder unter 10 Jahren. Falls ein inkorrekter Kurzschluss durch unterschiedliche Polaritäten verursacht, oder der Kondensator unter falschen Bedingungen verwendet wird, können heiße Oberflächen durch Bauteile auf der Platine entstehen.

Geisteraugen

Wie wäre es damit, zu Halloween den Kürbis noch etwas unheimlicher zu machen? Mit deinem KosmoDuino kannst du ihm Geisteraugen basteln.

DU BRAUCHST

- **KosmoDuino im Interaction Board**
- **2 Widerstände**
- **grüne LED**
- **rote LED**
- **3 Jumperkabel männlich-männlich**
- **4 Jumperkabel männlich-weiblich**
- **Breadboard**

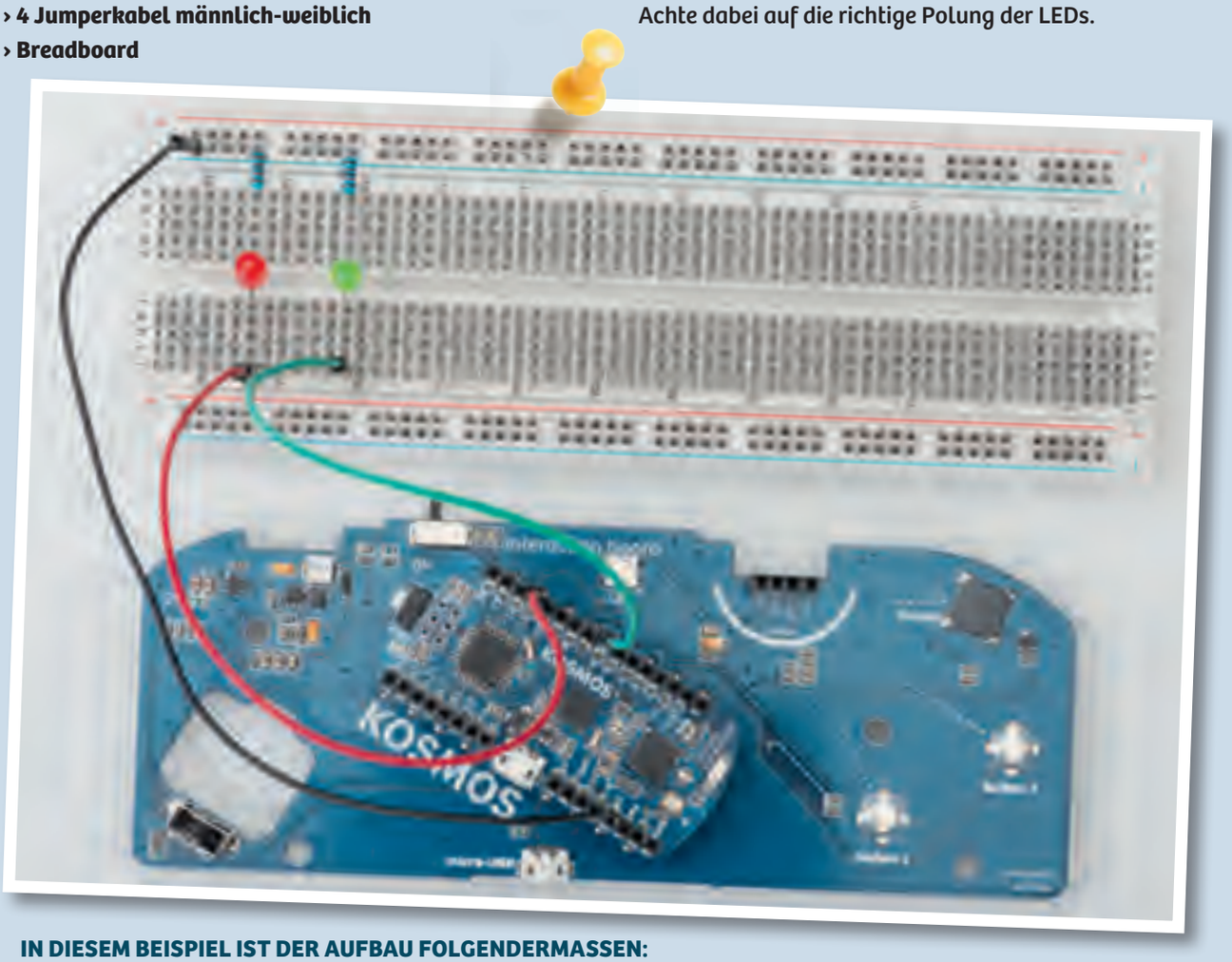
DER PLAN

Lass die Augen deines Kürbis zufällig bunt blinken. Dazu ertönt in unregelmäßigen Abständen ein gruseliges Rauschen.

VORBEREITUNG

Diesmal braucht es etwas mehr als nur das Interaction Board, denn du musst eine kleine elektronische Schaltung aufbauen. Die baust du aus den links genannten Teilen nach der Abbildung zusammen.

Achte dabei auf die richtige Polung der LEDs.



IN DIESEM BEISPIEL IST DER AUFBAU FOLGENDERMASSEN:

- Ein Jumperkabel verbindet einen Gnd-Pin mit der (-)-Leiste des Breadboards.
- Ein Jumperkabel verbindet Pin 11 des KosmoDuinos mit Feld j 57 auf dem Breadboard.
- Ein Jumperkabel verbindet Pin 6 des KosmoDuinos mit Feld j 51 auf dem Breadboard.
- Die Widerstände verbinden die (-)-Leiste und Feld a 57 sowie (-)-Leiste und Feld a 51.
- Das lange Beinchen der roten LED steckt in Feld f 57, das kurze in e 57. Bei der grünen LED verhält es sich entsprechend mit Reihe 51.
- Um die LEDs vom Breadboard zu den Kürbisaugen zu führen, verwendest du die 4 männlich-weiblich Jumperkabel um die LEDs an den entsprechenden Stellen mit dem Breadboard zu verbinden.

DAS PROGRAMM

Das Programm ist äußerst einfach. Zuerst werden Konstanten für die benutzten Pins definiert. In `setup()` geben wir dem Zufalssgenerator einen Startwert.

In der Hauptschleife wird zunächst mit `analogWrite()` die Helligkeit der jeweiligen LED auf einen zufälligen Wert, `random(0, 256)`, eingestellt. Das bewirkt das Flackern der Geisteraugen.

Um das Rauschen zu erzeugen, erzeugst du dann in der `if`-Anweisung mit `random(0, 1000)` wieder eine Zufallszahl. Diesmal zwischen 0 (kleister möglicher Wert) und 999 (größter möglicher Wert).
Ist die Zufallszahl größer als 900, dann schreibst du in der anschließenden `for`-Schleife mit `analogWrite(buzzerPin, random(0, 256));` 3000 Zufallswerte in den `buzzerPin`. Das ergibt das Rauschen im Lautsprecher.

Die Zufallszahlen, die mit Hilfe der `random()`-Funktion erzeugt werden, sind gleich verteilt. Das bedeutet, dass jede mögliche Zahl über lange Zeit betrachtet gleich häufig erzeugt wird. Das Rauschen wird aber nur in den Fällen ausgegeben, in denen die Zufallszahl größer ist als 900. Das passiert in ungefähr einem Zehntel der Fälle. Möchtest du, dass das Rauschen seltener auftritt, musst du statt der 900 eine größere Zahl wählen.

Du kannst diese blinkenden und flackernden Geisteraugen nun in einen Kürbis oder in ein selbst gebasteltes Gespenst einbauen.

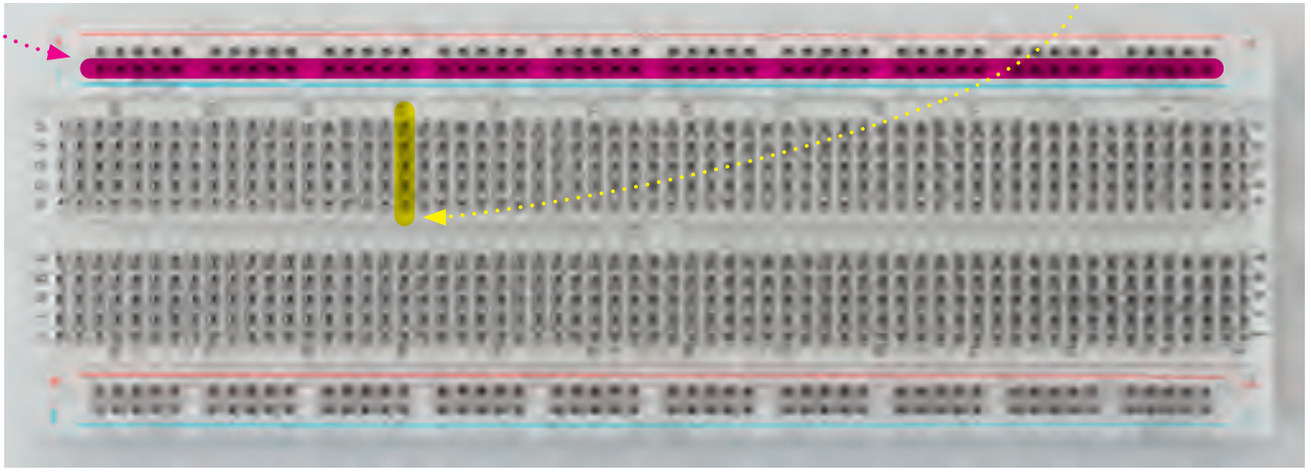
Die Geisteraugen lassen sich auch prima mit anderen Ideen kombinieren. Wie wäre es zum Beispiel mit einer Sirene, die aufheult, wenn du den Kürbis mit einer Taschenlampe anstrahlst? Sicherlich weißt du mittlerweile, wie du das machen kannst, und hast noch viel mehr tolle Ideen!

```
#include <KosmoBits_Pins.h>

const int buzzerPin = KOSMOBITS_BUZZER_PIN;
const int rotPin = 11;
const int gruenPin = 6;

void setup() {
  randomSeed(analogRead(3));
  pinMode(rotPin, OUTPUT);
  pinMode(gruenPin, OUTPUT);
}

void loop() {
  // Blinken
  analogWrite(rotPin, random(0, 256));
  analogWrite(gruenPin, random(0, 256));
  delay(200);

  // Rauschen
  if (random(0, 1000) > 900) {
    for (int i = 0; i < 3000; ++i) {
      analogWrite(buzzerPin, random(0, 256));
    }
  }
}
```

▲ *Geisteraugen*

BREADBOARD

Das Breadboard, selten auf Deutsch auch Steckbrett genannt, ist ein einfacher Weg, kleine elektronische Schaltungen aufzubauen. Die äußeren rot und blau gekennzeichneten Reihen sind dabei elektrisch miteinander verbunden. Hier legt man üblicherweise die Spannungsversorgung an. Dazu verbindet man einen der mit »Gnd« (englisch für Ground, deutsch Masse) gekennzeichneten Pins des Mikrocontrollers mit einem (-)-Feld, und einen »5V«-Pin mit einem (+)-Feld im Breadboard. Die senkrechten Spalten (a bis e und f bis j) sind ebenfalls miteinander elektrisch verbunden. Allerdings nur bis zu dem »Graben« in der Mitte des Breadboards. Die Details kannst du in der Abbildung erkennen.



Wie geht es weiter?

Du hast jetzt viele Grundlagen der Arduino-Programmierung erlernt. Darauf kannst und solltest du aufbauen. Das KosmoBits-Set erlaubt dir einen guten Start, denn neben dem eigentlichen Mikrocontroller hast du bereits ein Breadboard, einige LEDs, Jumperkabel und natürlich Sensoren. Mit Hilfe des Akkus kannst du das Ganze sogar unabhängig von einem Computer betreiben, ohne zusätzliche Teile zu benötigen. Der Akku ist sogar ziemlich leistungsstark. Das merkst du daran, dass er relativ selten aufgeladen werden muss. Mit anderen Worten: Du bist bestens ausgerüstet!

Im Internet wirst du jede Menge weitere Projektideen finden. Vermutlich wirst du dazu gelegentlich ein weiteres Elektronikteil benötigen. Den Rest hast du aber schon. Vor allem aber kannst du mittlerweile sogar schon programmieren.

Lass deiner Kreativität freien Lauf.

Viel Spaß dabei!

Eingerückt

Vielleicht hast du dich schon gewundert, dass viele Zeilen, die du in den Programmtexten hier findest, eingerückt sind, also ein wenig weiter rechts beginnen als andere. Die Antwort ist recht einfach: Das erleichtert den Überblick über den Programmtext.

Nehmen wir mal ein einfaches Beispiel:

```
if (digitalRead(taste1) == LOW) {
  rot = 255;
} else {
  rot = 0;
}
```

Das könntest du auch so schreiben:

```
if (digitalRead(taste1) == LOW) {
rot = 255;
} else {
rot = 0;
}
```

oder sogar noch schlimmer so:

```
if (digitalRead(taste1) == LOW) {rot = 255;} else
{rot = 0;}
```

◀ *Automatische Formatierung*

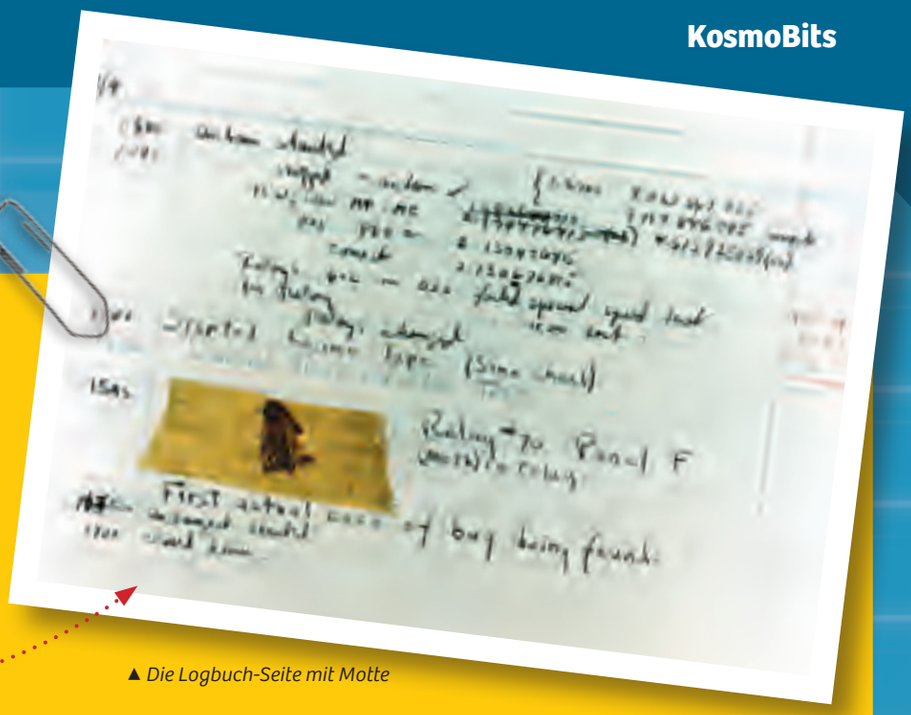
Das bedeutet alles genau das Gleiche, es ist nur etwas anders formatiert. Und trotzdem, im ersten Beispiel kannst du auf einen Blick die Struktur des Programms erkennen. Es ist leichter den Code zu lesen.

Deshalb hat es sich als praktisch erwiesen, den Programmcode jedes Mal eine Stufe weiter einzurücken, wenn ein neuer Block beginnt. Auch sollte in aller Regel jede einzelne Anweisung eine eigene Zeile bekommen.

Die Arduino-Umgebung unterstützt dich dabei und rückt neue Blöcke in den meisten Fällen von alleine ein. Wenn du Textteile doch einmal per Hand einrücken möchtest, dann benutze dafür möglichst nicht die Leertaste, sondern die Tab-Taste.

NACHGEHAKT

Was ist eigentlich ein Bug?

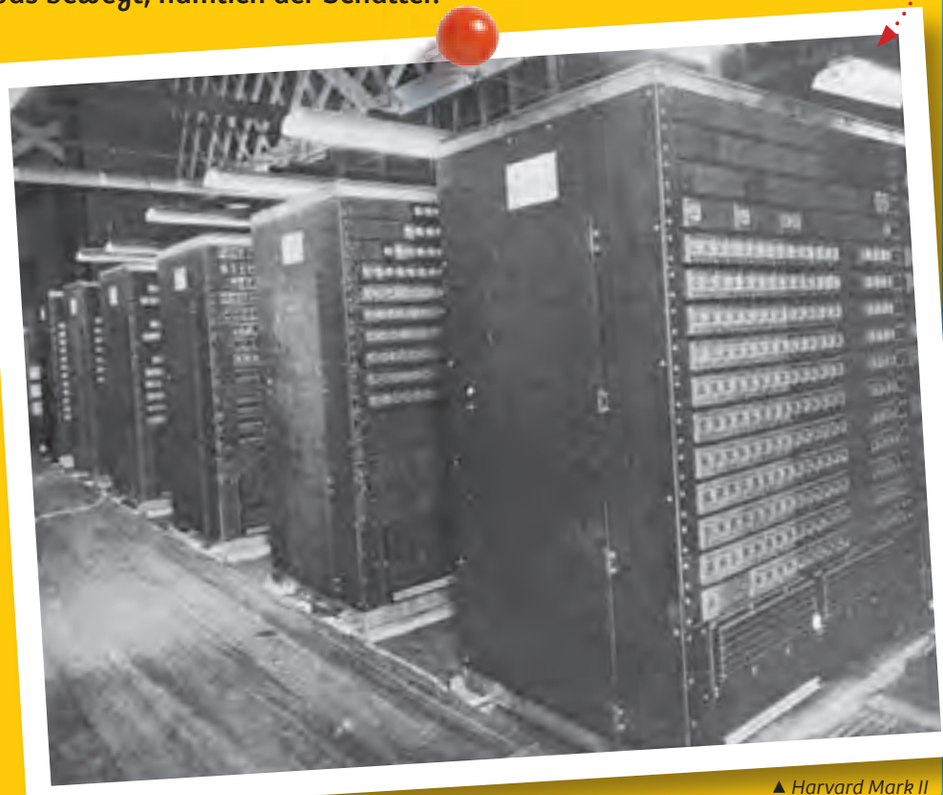


▲ Die Logbuch-Seite mit Motte

Wenn ein Programm nicht das tut, was es eigentlich soll, spricht man von einem *Bug*. Man könnte es auch einfach einen Fehler nennen, der englische Begriff *Bug* (deutsch: Käfer) wird aber viel öfter verwendet. Die Geschichte dahinter ist erstaunlich!

Frühere Computer, wie zum Beispiel der 1947 fertig gestellte *Harvard Mark II*, besaßen noch keine Mikroprozessoren. Stattdessen rechneten sie mit Hilfe von elektro-mechanischen Relais. Ein Relais besteht im Grunde aus einem Elektromagneten und einem Schalter, der sich magnetisch betätigen lässt. Schaltet man den Elektromagneten an, betätigt er den Schalter. Schaltet man den Magneten wieder an, bewegt sich der Schalter wieder in die »Aus«-Stellung. Wichtig ist, dass sich in einem solchen Relais tatsächlich etwas bewegt, nämlich der Schalter.

Am 9. September 1947 machten sich die für den Mark II zuständigen Techniker auf die Suche nach einem Fehler. Sie fanden ihn: Es handelte sich um eine Motte, die sich in einem Relais verklemmt hatte. Der erste *Bug* in einem Programm war also tatsächlich ein Insekt. Die Motte haben sie pflichtbewusst in das Logbuch des Mark II eingeklebt.



▲ Harvard Mark II

Auf einer Maker Faire ►

Arduino und die Maker

▲ *Workshop auf einer Maker Faire*

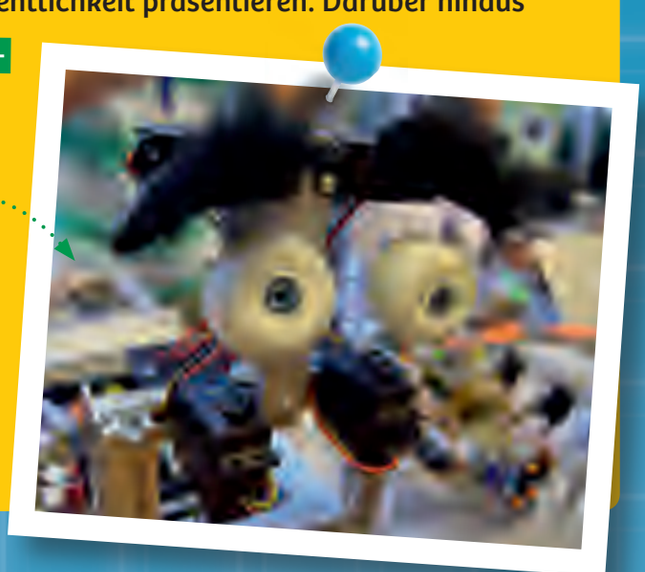
Hast du auch schon einmal davon geträumt, ein richtiger Erfinder oder eine Erfinderin zu sein? Neue Maschinen und Apparate zu erdenken und sie dann Wirklichkeit werden zu lassen? Immer mehr Menschen geben sich nicht mehr mit Träumen zufrieden. Sie machen es einfach. Sie sind *Maker*!

Maker ist englisch und bedeutet »Macher«. Früher hätte man diese Macher einfach Tüftler oder Bastler genannt. Doch das trifft es nicht ganz: Während die meisten Tüftler früher eher alleine in ihrer Bastelstube vor sich hin gewerkelt haben, zieht es die Maker auch gerne in die Öffentlichkeit.

So sind in vielen Städten sogenannte Hackerspaces oder Makerspaces entstanden. Sie sind eine Mischung aus Treffpunkt und Gemeinschaftswerkstatt. Hier tauschen sich die Maker aus, präsentieren ihre neuesten Ideen, geben sich Ratschläge und können dabei auf gemeinschaftliche Teile und Werkzeuge zurückgreifen.

Weltweit entstehen auch immer mehr ***Maker Faires***. Das ist wieder englisch und bedeutet »Maker Messe«. Hier können sich die Maker einer breiteren Öffentlichkeit präsentieren. Darüber hinaus gibt es dort meist ein umfangreiches Angebot an **Workshops**, in denen erfahrene Macher dazulernen und Einsteiger die ersten Schritte erlernen können.

Eine wichtige Rolle spielen bei den **Projekten** der Maker oft Mikrocontroller wie der Arduino. Erst durch sie ist es möglich geworden, komplizierte Apparate zu verwirklichen, ohne dafür erst teure Spezialelektronik entwickeln zu müssen.



Maker-Projekt ►

◄ Ein 3D-Drucker bei der Arbeit

3D-Drucker

Was nützt die beste Elektronik mit der tollsten Software, wenn man für seine Projektidee einfach kein passendes Gehäuse findet? Mittlerweile gibt es auch dafür eine passende Maker-Lösung: Die 3D-Drucker.

Das sind Geräte, die nicht zweidimensional (2D) auf Papier drucken, sondern dreidimensional (3D) Dinge und Gegenstände ausdrucken können. Klingt nach Science-Fiction, ist aber vom Prinzip eigentlich doch ganz einfach. Man benutzt dazu meist ein Kunststoffmaterial, das zunächst ähnlich wie ein Faden auf einer Spule aufgerollt ist. Dieses *Filament* wird dann in einer Düse so weit erhitzt, dass es schmilzt. In geschmolzener Form wird der Kunststoff computergesteuert an einer bestimmten Stelle wieder ausgespritzt. Der Kunststoff kühlt dann ab und härtet aus. Tropfen für Tropfen, Schicht für Schicht werden so aufwändigste Gegenstände aufgebaut.

Geräte, die so etwas können, gibt es schon länger, aber erst in den letzten Jahren sind sie für Hobby-Bastler bezahlbar geworden.

Übrigens: Auch die ersten Test-Versionen des KosmoBits-Controllers sind auf diese Weise entstanden!

▲ Filament für 3D-Drucker

◄ Verschiedene 3D-gedruckte Prototypen aus der KosmoBits-Entwicklung

Häufige Fehlermeldungen

`error: expected ';' before ')'`

Es wurde ein Semikolon vor einer schließenden Klammer erwartet, aber nicht gefunden.

BEISPIEL:

```
In function 'int mittelwert2()':
DoppelKlatscher:86: error: expected ';' be-
fore ')' token
  for (int i = 0; i < N, ++i) {
                         ^
exit status 1
expected ';' before ')' token
```

LÖSUNG:

Nach `i < N` muss das Komma durch ein Semikolon ersetzt werden.

`error: '...' was not declared in this scope`

Ein im Programm benutzter Name, ist dem Compiler nicht bekannt. Oftmals ein Zeichen für einen einfachen Tippfehler.

BEISPIEL:

```
error: 'pinnMode' was not declared in this
scope

  pinnMode(A1, INPUT);
```

LÖSUNG:

Tippfehler korrigieren: Richtig heißt es `pinMode`.

`error: '...' does not name a type`

Ein im Programm benutzter Typ ist dem Compiler nicht bekannt. In den meisten Fällen handelt es sich entweder um einen Tippfehler, oder du hast vergessen eine entsprechende Datei einzubinden.

BEISPIEL:

```
error: 'KosmoBits_Pixel' does not name a
type
  KosmoBits_Pixel pixel;
```

LÖSUNG:

Binde die »KosmoBit_Pixel.h« ein:

```
#include <Adafruit_NeoPixel.h>
#include <KosmoBits_Pixel.h>
```

```
avrdude: ser_open(): can't open device "\\.\COM4": Zugriff verweigert
```

Vermutlich hast du vergessen, den seriellen Monitor zu schließen, bevor du dein Programm hochladen wolltest.

LÖSUNG:

Schließe den seriellen Monitor und lade das Programm erneut hoch.

```
avrdude: ser_open(): can't open device "\\.\COM4": Das System kann die
angegebene Datei nicht finden.
```

1. Du hast vergessen, den KosmoDuino mit dem Computer zu verbinden. Vielleicht hast du das USB-Kabel auch nur versehentlich in die Ladebuchse des Interaction Boards gesteckt.

LÖSUNG:

Verwende das USB-Kabel um deinen Computer mit dem USB-Port am KosmoDuino zu verbinden.

2. Es ist der falsche Port eingestellt. Überprüfe das im Menü »Werkzeuge → Port« und ändere den Port gegebenenfalls.

LÖSUNG:

Wähle einen anderen Port aus.

```
Wenig Arbeitsspeicher verfügbar, es können Stabilitätsprobleme auftreten.
```

```
Nicht genug Arbeitsspeicher
```

Dein Programm füllt fast den gesamten Arbeitsspeicher des Mikrocontrollers oder ist sogar schon zu groß dafür. Vermutlich hast du ein ziemlich großes Array angelegt.

BEISPIEL:

```
int array[1000] = {}; // Das ist zu groß!
```

LÖSUNG:

Ändere dein Programm so, dass du mit einem kleineren Array auskommst.

NOTIZEN

››› KosmoBits Quickstart-Guide

MIT SPIELEN LOSLEGEN:

› Bau dein eigenes Gamepad zusammen
- Detaillierte Anweisungen findest du in der Anleitung ab S. 3

› Lade den Akku des Gamepads
- Verwende dazu den USB-Anschluss am Interaction Board, nicht den Anschluss am Kosmoduino!

› Lade die KosmoBits-App auf dein Tablet oder Smartphone
- Die App findest du bei Google Play oder im Apple App Store

› Aktiviere Bluetooth auf deinem Tablet oder Smartphone

› Starte die App

› Schalte dein Gamepad an
- Schiebe den Schalter am oberen Rand des Gamepads auf die Position »on«
- Die Verbindung erfolgt automatisch

› Bau die SensorBots zusammen und halte sie bereit

› Viel Spaß beim Spielen!

MIT PROGRAMMIEREN LOSLEGEN:

› Installiere die Arduino-Software auf deinem Computer
- Die Software bekommst du kostenlos unter **arduino.cc/en/Main/Software**. Wir empfehlen die von uns getestete Version 1.6.5

› Öffne die Arduino-Software

› Schreibe dein Programm in der Arduino-Software

› Verbinde den KosmoDuino-Microcontroller per USB mit deinem PC

› Lade dein Programm auf den KosmoDuino
- Dein Programm wird direkt nach dem Hochladen vom KosmoDuino ausgeführt
- Viele spannende Programmier-Projekte findest du in der Anleitung
- Für viele Programmier-Projekte benötigst du die KosmoBits-Bibliotheken. Diese kannst du kostenlos unter **kosmobits.de/downloads** herunterladen. (Detaillierte Anweisungen findest du in der Anleitung auf S. 8)

› Viel Spaß beim Programmieren!

ÜBERSICHT ÜBER EINIGE WICHTIGE BEFEHLE

PROGRAMMSTRUKTUR	ERKLÄRUNG
`void setup(){ }`	Initialisierung. Am Anfang jedes Programms können erste Einstellungen vorgenommen werden, das sogenannte »setup«. Hier wird zum Beispiel festlegt, welche Pins welche Funktion übernehmen sollen oder wo die LEDs angeschlossen sind. Das setup wird nach jedem Start oder Neustart nur einmal durchgeführt.
`void loop(){ }`	Hauptschleife. Der Hauptteil des Programms ist der sogenannte Loop. Es ist eine Schleife, die sich solange wiederholt, wie der Microcontroller läuft. Hier sind alle wichtigen Funktionen untergebracht, die dauerhaft das System überwachen oder verändern.

VARIABLEN	BEISPIEL
`int` (ganze Zahlen)	`int led = 13;`
`long` (lange ganze Zahlen)	`long meinWert = 111111;`
`float` (Fließkommazahlen)	`float meinWert = 2.5;`
`char` (Alphanumerische Zeichen: Buchstaben, Zahlen, Sonderzeichen)	`char meinBuchstabe = ´a´;`
`array` (Variablenfeld)	`int meineWerte[4] = {12, 18, 23, 86}`

BEFEHL	BEISPIEL	ERKLÄRUNG
`pinMode()`	`pinMode(led, OUTPUT);`	Setzt den Pin mit dem Namen »led« als Ausgabe-Pin fest.
`digitalWrite()`	`digitalWrite(led, HIGH);`	Legt einen Strom auf dem Pin mit dem Namen »led« an.
`digitalRead()`	`digitalRead(Schalter)`	Liest den Zustand von Pin »Schalter« aus: Ist der Schalter an (HIGH) oder aus (LOW)?
`analogWrite()`	`analogWrite(led, 255)`	Setzt den Pin mit dem Namen »led« auf den Wert 255
`analogRead()`	`analogRead(temp)`	Liest den Wert, den der Pin mit dem Namen »temp« liefert.
`delay()`	`delay(1000);`	Lässt das Programm eine Sekunde warten (=1000 Millisekunden).
`Serial.begin()`	`Serial.begin(115200)`	Beginnt die Übertragung von Daten zum PC mit 115200 Baud (Symbolen pro Sekunde)
`Serial.println()`	`Serial.println(„Hallo“);`	Zeigt im seriellen Monitor »Hallo« an.