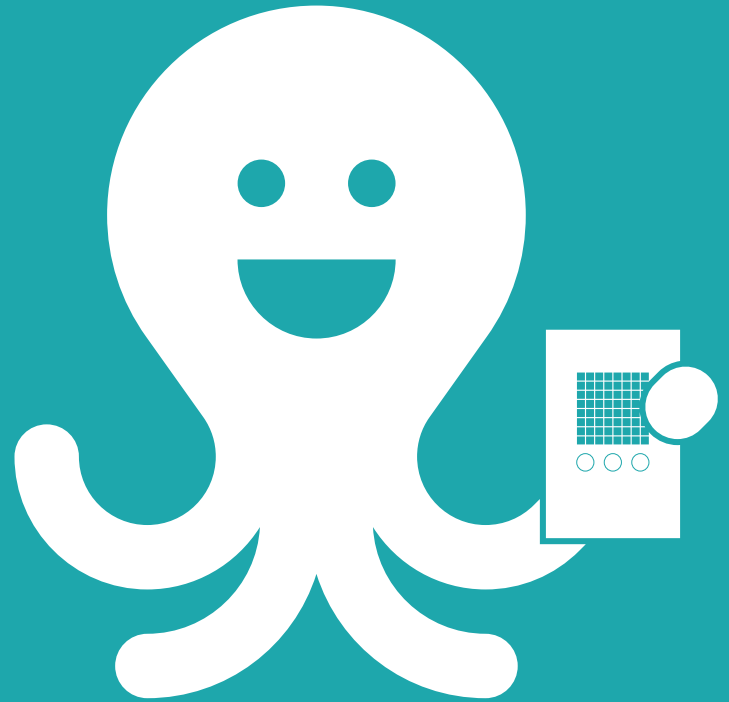


PROGRAMMIEREN LERNEN MIT DER OXOCARD

PROGRAMMIERKURS FÜR KIDS



DESIGNED                     
IN-LIEBEFELD                                        
MADE WITH PASSION

MADE BY **OXON**®

Inhaltsverzeichnis	2	Jetzt wird's laut	80
Einleitung	4	Wie Töne entstehen	82
Für wen und was ist dieses Büchlein?	6	Die Sireenschleife	85
Gemeinsam macht's mehr Spass	8	Spiel mir ein Lied	88
Was ist ein Computer?	10	Das Spiel mit der Beschleunigung	92
Was ist eine Programmiersprache?	14	Wo ist oben?	94
Was ist die OXOcard?	18	Gravitation	96
Vom Häuschenpapier auf den Computer	22	Zusammen macht's mehr Spass	102
Es geht los. Wir schreiben unser erstes Programm!	24	Kommunikation ist alles	104
Wie zeigt der Computer Bilder an?	32	„Ich bin die OXOcard 1 und ich bin hier!“	106
Wir versuchen als erstes, ein Pixelbild zu zeichnen	34	Das wiederverwendbare Herz	112
Wir basteln uns eine Smiley-Maschine!	40	Spiel – „Meteorit“	116
Zeichnen mit Grafik-Befehlen	46	Basisversion des Meteor-Games	122
X- und Y-Koordinaten	48	Erweiterungen	126
Grafikbefehle bringen's	50	Gib mir einen Beep!	126
Darf es etwas weniger hell sein?	54	Schneller!	127
Let's move!	56	Spiel-Intro	128
Rechnen	58	Eigene Crash-Animation	130
Variablen und weitere If-Anweisungen	61	Was kommt als Nächstes?	132
Jetzt animieren wir mal was	64	Glossar	136
Zufällige Begegnungen	72	Alle Oxocard-Befehle auf einen Blick	140
Die Zufallsfunktion	74	Konstanten der Tonleiter	152
Zufällige Kunst	78	Lösungen	156

FÜR WENN UND WAS IST DIESES BÜCHLEIN?

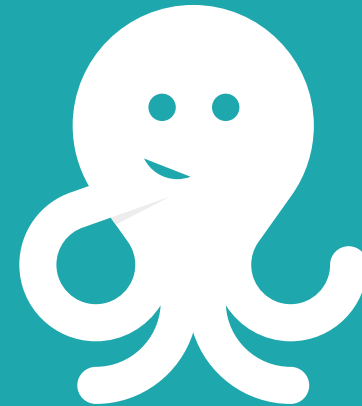
Mit diesem Büchlein möchten wir dir die spannende Welt des Programmierens näherbringen und dein Interesse wecken, einmal hinter die Kulissen deines Computers oder Handys zu schauen. Hast du dir schon einmal überlegt, wie das alles eigentlich genau im Hintergrund funktioniert? Wie programmiert man beispielsweise ein Spiel oder wie erkennt das Handy Neigebewegungen, wie sie bei vielen Apps heute genutzt werden? All dies und noch viel mehr lernst du hier. Unser Ziel ist es, dir zu zeigen, dass Programmieren Spass macht. 😊

Wenn du programmieren kannst, eröffnen sich dir viele Möglichkeiten. Du kannst damit nicht nur Spiele programmieren – wobei dies wohl aktuell das spannendste Thema für dich sein dürfte 😊 – sondern du kannst damit Computer so einrichten, dass sie Aufgaben für dich verrichten. So kannst du beispielsweise einen Computer dazu animieren, Hausaufgaben für dich zu schreiben. Aber halt! Soweit wollen wir nun definitiv nicht gehen. 😊

Programmieren wird in Zukunft sehr wichtig werden, wir glauben sogar, dass es ähnliche Bedeutung wie Lesen und Schreiben haben wird. Daher möchten wir dich dafür begeistern und dir mit ein paar einfachen Beispielen zeigen, dass es nicht viel braucht, Computer so zu programmieren, dass sie Dinge für dich verrichten.

Die OXOcard und dieses Begleitbüchlein sind in erster Linie für Schülerinnen und Schüler gedacht und können sowohl im MINT-Unterricht als auch im Selbststudium genutzt werden.

UNSER ZIEL IST ES, DIR
ZU ZEIGEN, DASS PROGRAMMIEREN
SPASS MACHT.

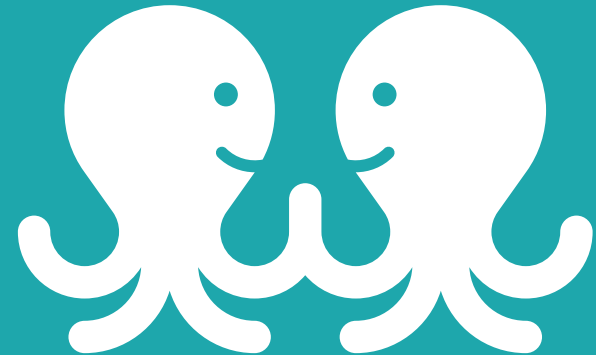


GEMEINSAM MACHTS MEHR SPASS

Du kannst diese Anleitung selbstständig durcharbeiten, allerdings empfehlen wir dir, dies zu zweit mit einer Kollegin oder einem Kollegen zu machen. In der Programmierwelt gibt es eine verbreitete Arbeitsmethodik, die sich Pairprogramming oder zu deutsch „Paar-Programmierung“ nennt. Damit ist kein Liebespaar gemeint, sondern ein Team aus zwei Personen, die sich gegenseitig helfen um rascher und mit weniger Fehlern zum Ziel zu kommen. Pairprogramming wird vor allem dann genutzt, wenn eine Aufgabe besonders schwierig ist und man sich beim Programmieren mit jemanden austauschen will. Für uns hat es eine etwas andere Aufgabe. Wie du bald sehen wirst, musst du beim Programmieren ganz exakt sein. So braucht unsere Programmiersprache nach jedem Befehl beispielsweise einen Strichpunkt „;“, es gibt runde (), geschweifte {} und eckige [] Klammern, die ganz exakt angewendet werden müssen. Der Computer toleriert keine Fehler. Die Befehle müssen ebenfalls genau so eingegeben werden, wie der Computer dies wünscht, mit korrekter Gross- und Kleinschreibung. Beim Diktat gibt es bei Fehlern Abzüge und eine schlechtere Note, beim Programmieren funktioniert es einfach nicht. Da hier gerade zu Beginn das Frustrationspotential rasch hoch sein kann, sind vier Augen Gold wert. 😊

Such dir also einen Mitschüler oder eine Mitschülerin – ihr könnt auch zu dritt. Eine Person programmiert jeweils, der oder die andere/n schauen ihm/ihr über die Schulter und stellen sicher, dass sich kein Fehler einschleicht. Dann wechselt ihr ab. So macht es noch mehr Spass und ihr kommt rascher voran. 😊

ZUSAMMEN MACHTS NOCH MEHR SPASS UND IHR KOMMT RASCHER VORAN.



WAS IST EIN COMPUTER?

Du hast sicherlich schon viele persönliche Erfahrungen mit Computern gemacht. Diese sind heute allgegenwärtig und nicht mehr aus unserem Leben wegzudenken. Neben den Geräten, die du sofort als Computer wahrnimmst, wie das Notebook, ein Tablet oder ein Handy, gibt es mittlerweile eine ganze Menge an Geräten, die nicht sofort als Computer erkennbar sind, aber trotzdem einen eingebaut haben. Wenn du morgens aus dem Haus gehst, hast du bereits mehrere „Computer“ gebraucht. Du wurdest vielleicht geweckt durch einen Radiowecker, dann hast du warm geduscht – der elektronischen Heizung sei Dank – und deine Mutter hat dir auf dem Induktionsherd einen Kakao zubereitet. Du hast ein blinkendes Velolicht? Die Blinkgeschwindigkeit regelt vermutlich ein ganz kleiner Computer. Letztlich wird in den meisten Schulen heute die Schulglocke mit einer elektronischen Zeitschaltuhr gesteuert – wieder ein Computer.

AUFGABE

Was für Computer begegnen dir während einem normalen Wochentag? Schreib dir auf, was alles deiner Meinung nach einen Computer enthält. Diskutiere die Liste mit deinen Mitschülern.



LÖSUNG - SEITE 158

Du hast eine eindrückliche Liste von Computern zusammengestellt? Das ist erst der Anfang. Wir werden in Zukunft noch viel mehr Computer versteckt in allerlei Alltagsgegenständen sehen. Beängstigt dich das ein wenig? Keine Angst, Computer schüchtern nur ein, solange man nicht weiss, was dahintersteckt und durch Film und Fernsehen schaurige Geschichten über Computerintelligenzen und gewiefte Hacker erzählt werden.

Unser Ziel ist es, dir zu zeigen, dass alles eigentlich ganz einfach zu verstehen ist und du selber in einfachen Schritten solche Maschinen programmieren kannst.

Wieso heisst der Computer Computer?

Das Wort „Computer“ stammt vom englischen „to compute“ ab, also dem Verb „berechnen“. Dies war auch die erste Aufgabe von Computern, Dinge zu berechnen.

Die ersten Computer

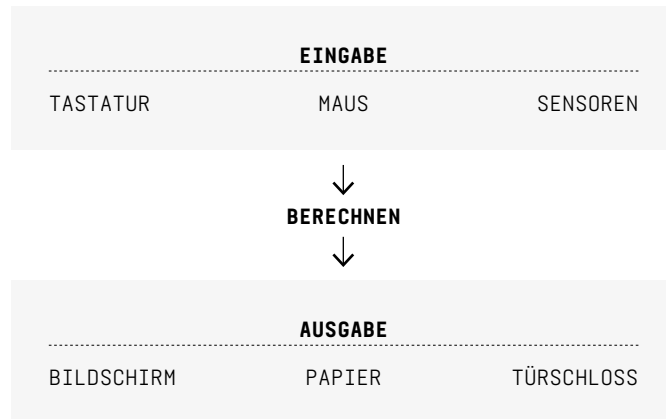
Zu Zeiten des zweiten Weltkriegs hat Alan Turing auf Seiten der Alliierten mittels erster primitiver Rechenmaschinen versucht, das damals gefürchtete Verschlüsselungsgerät Enigma der Deutschen zu dechiffrieren. Turing war ein ausserordentlich intelligenter Mathematiker und ihm gelang das Unglaubliche. Mit seiner programmierbaren Maschine konnten wichtige Meldungen des deutschen Militärs im Geheimen mitgelesen werden, was letztlich entscheidend war, um die Terrorherrschaft Hitlers zu beenden.

Charles Babbage, Konrad Zuse, Alan Turing und viele andere Mathematiker und Erfinder haben an der Entwicklung von Computern gearbeitet. Alles begann Mitte des neunzehnten Jahrhunderts und ungefähr ab 1980 startete mit der Massenproduktion von Computerchips der Durchbruch und Computer wurden zu günstiger Massenware. Vermutlich

hat dein blinkendes Velolicht mehr „Computerpower“, als die ersten verkauften Computer dieser Gründerzeit sie hatten. Die Entwicklung punkto Leistung ist enorm und es ist noch kein Ende in Sicht.

Doch was ist jetzt eigentlich genau ein Computer? Du wirst auf dem Internet verschiedene Beschreibungen dazu finden. Wir möchten ihn so umschreiben: Ein Computer ist eine sehr ausgefeilte elektronische Maschine, die akribisch exakt Befehle ausführt, die zuvor festgelegt, d.h. „einprogrammiert“ wurden.

Wenn wir eine Maschine bauen, soll diese einen Zweck erfüllen, daher muss sich irgendetwas tun, d.h. ein Ergebnis erzeugen. Der PC hat dafür beispielsweise einen Bildschirm, auf dem Bilder und Texte angezeigt werden. Es gibt aber auch Computer, die nur Lämpchen blinken lassen, wie beispielsweise dein Velolicht, oder Schalter umstellen, wie bei der Heizung. Trotzdem sind dort auch Computer am Werk.



Damit ein Computer auf seine Umwelt reagieren kann, braucht es neben der Ausgabe noch etwas Zweites: Die Eingabe. Wenn wir etwas starten oder schreiben möchten, brauchen wir Tastatur und Maus, um dies der Maschine mitteilen zu können. Über diese „Sinne“ teilen wir dem Gerät etwas mit. Es gibt auch Computer, die keine Tastatur haben, beispielsweise ein Backofen, der nur zwei Drehregler für Temperatur und Heizprogramm hat. Und dann gibt es welche, die einen anderen „Input“ erwarten: Eingaben von Sensoren. Beispielsweise schaltet der Computer des Treppenlichts dieses ein, sobald ein Mensch vorbeiläuft. In dem Fall ist ein Infrarotsensor installiert, der die Körperwärme von Säugetieren – und damit auch Menschen – erkennt und dies dem Computer mitteilt.

AUFGABE

In einem modernen Smartphone ist ein sehr leistungsfähiger Computer eingebaut. Dieser hat eine Vielzahl von Sinnen, d.h. über seine Sensoren nimmt er die Umgebung wahr. Kannst du sie alle benennen? Schreibe eine Liste und schreibe dazu, was der Computer damit alles erkennen kann.

LÖSUNG - SEITEN 159 - 161



WAS IST EINE PROGRAMMIERSPRACHE?

Eine überraschende Geschichte zu den Anfängen der Programmierung.

Die ersten Computer waren zu Beginn mechanisch gebaut, erst viel später, mit der Erfindung des Transistors, konnte man anstelle von mechanischer Kraft auch Elektrizität nutzen. Einen der ersten Computer hatte der Engländer Charles Babbage entwickelt. Und wer hat das erste Programm dazu geschrieben? Baroness Ada Lovelace! Ja, du liest richtig, von wegen, Frauen interessieren sich nicht für Computer. Nach ihr wurde sogar die Programmiersprache Ada benannt. Babbage hat seinen Computer übrigens nur „theoretisch“ gebaut; es gab nie eine funktionierende Maschine davon. Und Ada hat ihr erstes Programm daher auch nur auf Papier verewigt und im Kopf durchgespielt.

Wir brauchen Sprachen, um mit anderen zu kommunizieren. Obwohl eine Programmiersprache grundsätzlich wenig mit einer gesprochenen Sprache zu tun hat, dient sie einem ähnlichen Zweck. Man will in dem Fall einer Maschine mitteilen, was sie zu tun hat. Die Maschine spricht aber nicht Englisch oder Deutsch.

Was denkst du, welche Sprache spricht denn nun eigentlich ein Computer? Es ist eine Sprache, die nur aus Nullen und Einsen besteht und für einen Menschen kaum „zu sprechen“ ist. Man nennt sie Maschinensprache.

EXPERIMENT

Wir basteln uns eine eigene kleine Maschinensprache. Hierzu setzt ihr euch zu zweit zusammen. Du bist der Programmierer und dein Kamerad/deine Kameradin ist die Maschine.

Deine Maschine kann nur zwei Dinge unterscheiden: tippen auf die linke (0) oder die rechte (1) Schulter. Wir definieren folgende Maschinensprache:

0 0, d.h. zweimal links Tippen, bedeutet, dass die Maschine einen Schritt vorwärts geht. Hier die gesamte Liste:

0 0

EINEN SCHRITT VORWÄRTS

1 1

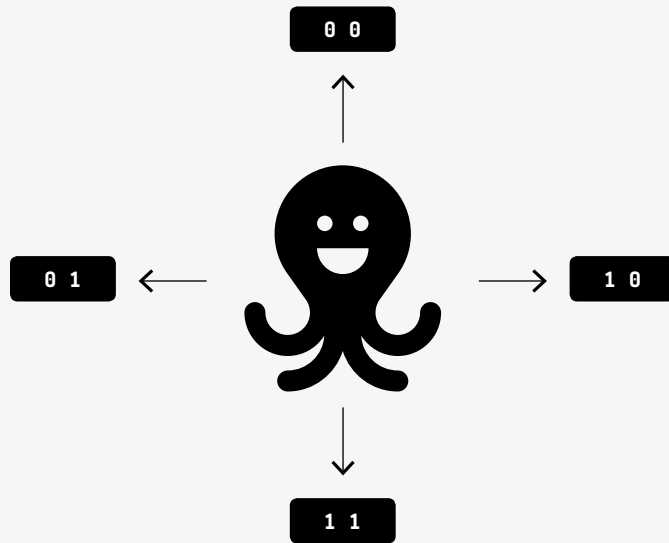
EINEN SCHRITT RÜCKWÄRTS

0 1

NACH LINKS DREHEN

1 0

NACH RECHTS DREHEN



Wir schreiben jetzt ein kleines „Maschinenprogramm“:

0 0 - 0 1 - 0 0 - 0 1 - 0 0 - 0 1 - 0 0

„Tippe“ dies jetzt mal in deine Maschine ein und schau, was passiert. Wenn ihr es einmal durchgespielt habt, wechselt ab, so dass du einmal eine Maschine bist und dein Kamerad/ deine Kameradin der Programmierer/ die Programmiererin.

Denke dir einen eigenen Weg aus und schreibe ihn dir Maschinsprache auf. Du kannst das Experiment auch alleine durchführen, indem du auf einem Blatt Häuschenpapier mit dem Stift jeweils ein Häuschen vorwärts oder rückwärts fährst.

Du wirst gemerkt haben, dass es schnell kompliziert wird und du vor lauter Nullen und Einsen nicht mehr weisst, was du eigentlich befehlen wolltest. Hierbei hatte das obige Beispiel pro Befehl nur zwei Stellen, die man im Computerefachjargon übrigens „Bits“ nennt. Mit zwei Bits kann man maximal vier Dinge mitteilen: **00**, **01**, **10** und **11**. Mit 8 Bits hat man bereits 256 Möglichkeiten. Probier's aus!

Interessant und doch recht überraschend ist die Tatsache, dass jeder heute kaufbare Computer intern ausschliesslich mit Nullen und Einsen arbeitet und dein Handy oder dein Computer demonstrieren eindrucklich, dass damit wirklich fast alles möglich ist.

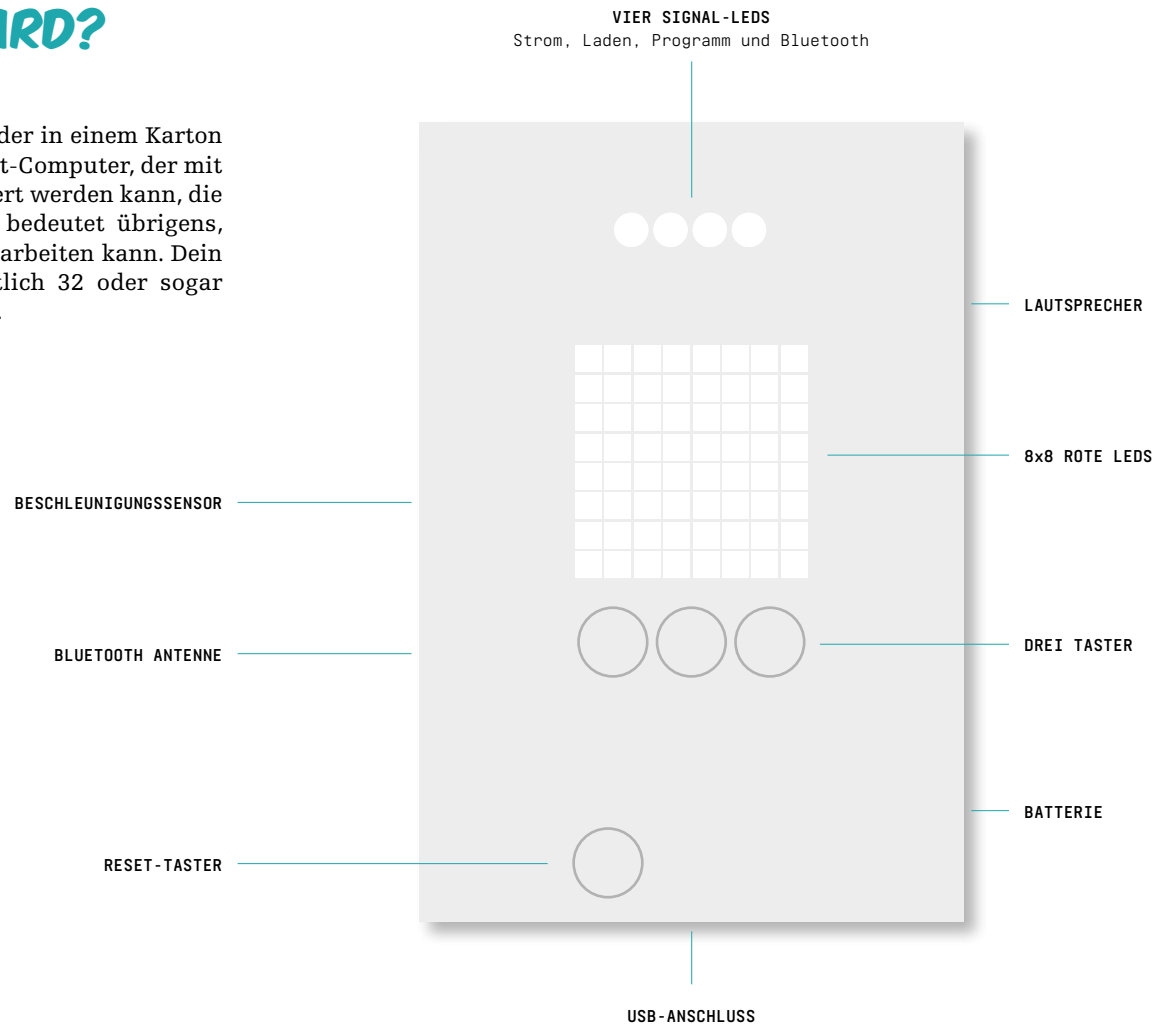
Man hat relativ rasch aufgehört, mit Bits zu programmieren, da die Fehlerquote einfach viel zu hoch war und man innert kurzer Zeit nicht mehr wusste, was man eigentlich wollte. Die ersten Programmierer haben daher diese Bit-Folgen zu Befehlen zusammengefasst und diesen passende Namen gegeben. Damit waren die ersten sogenannten Assembler-Programmiersprachen erfunden.

Wenn wir beim obigen Beispiel bleiben wollen, können wir zum Beispiel in unserer Programmiersprache den Befehl „Dreh im Kreis“ formulieren. Ich schreibe also „Dreh im Kreis“, meine aber: **0 0 - 0 1 - 0 0 - 0 1 - 0 0 - 0 1 - 0 0** gecheckt? Die Programmiersprache erlaubt es mir, in meiner Sprache Befehle aufzuschreiben und ein Übersetzungsprogramm übersetzt dies dann in die Maschinsprache.

Das war doch nicht so schwer, oder? Du hast jetzt die Grundprinzipien eines Computers kennengelernt und bereits ein bisschen Programmierluft geschnuppert. Wie Ada Lovelace hast du dafür Papier und Bleistift verwendet. Jetzt wollen wir aber mit den Experimenten beginnen.

WAS IST DIE OXOCARD?

Die OXOcard ist ein kleiner Computer, der in einem Karton verpackt ist. Darin enthalten ist ein 8bit-Computer, der mit einer Programmiersprache programmiert werden kann, die C++ (sprich: C-plus-plus) heisst. 8bit bedeutet übrigens, dass der Computer 8bits auf einmal bearbeiten kann. Dein Computer im Smartphone hat vermutlich 32 oder sogar 64bits und ist damit deutlich schneller.



Das steckt in der OXOcard

Der Computer der OXOcard heisst ATmega328p und hat eine Taktfrequenz von 8 Megahertz. Diese Zahl gibt an, wie schnell der Computer seine Maschinenbefehle abarbeiten kann. Bei diesem Chip braucht ein Befehl in der Regel zwei Takte. 8 Megahertz bedeutet, dass er 8 Millionen Takte pro Sekunde (!) beziehungsweise 4 Millionen seiner Maschinenbefehle in jeder Sekunde verarbeiten kann. Dies klingt auf den ersten Blick nach sehr viel. Man muss aber beachten, dass die Maschinenbefehle sehr primitiv sind und es ein- bis zweitausende solcher Befehle braucht, um z.B. eine E-Mail von einem PC aus zu verschicken. Unser Chip ist also trotz der Riesengeschwindigkeit im Verhältnis zu anderen Computern gemütlich unterwegs. Für unsere Zwecke reicht er aber allemal.

Die OXOcard hat folgende Ausgabemöglichkeiten:

- Eine 8x8 grosse LED-Fläche („LED-Matrix“ genannt), d.h. total 64 Leuchtdioden, mit denen du Bilder, Texte und Animationen darstellen kann. Die LEDs dienen uns also als Pixel zum Zeichnen.
- Einen Lautsprecher, mit dem du Töne erzeugen kannst („Piezo“ genannt).
- Vier LED-Lämpchen im oberen Bereich, die dir den Status der Karte anzeigen.
- Eine Bluetooth-Antenne, mit der du mit anderen Karten oder einem Computer bzw. Handy über Funk kommunizieren kannst.

Folgende Sensoren stehen dir zur Verfügung:

- Drei Taster unterhalb der LED-Matrix, sowie einen etwas versteckten Reset-Taster.
- Einen Beschleunigungssensor, den du für verschiedene Experimente nutzen kannst.
- Mit der Bluetooth-Antenne kannst du natürlich auch Meldungen anderer Karten oder Computer einfangen, wodurch diese auch zu einem Sensor wird.

Die Karte enthält zudem je nach Modell einen Akku, den du einfach über die USB-Buchse an einem Computer oder einem Handyladegerät laden kannst.

Die OXOcard hat ein eingebautes Testprogramm, das du nun sofort starten kannst.

Falls es nicht funktioniert, ist der eingebaute Akku leer oder dieser ist nicht installiert. In dem Fall steckst du die Karte mit einem USB-Kabel entweder an einen Computer oder ein Handy-Ladegerät.

Nun möchtest du sicher wissen, wie du solche coolen Dinge selber programmieren kannst? In den nächsten Kapiteln wirst du schrittweise lernen, wie man auf dem Computer Bilder und Animationen erzeugt, und wie du die OXOcard dazu bringen kannst, Töne wiederzugeben. Später zeigen wir dir, wie du mit deiner Karte über Funk mit anderen Karten kommunizieren kannst.

VOM HÄUSCHENPAPIER AUF DEN COMPUTER



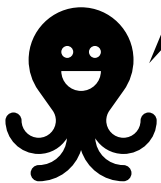
ES GEHT LOS. WIR SCHREIBEN UNSER ERSTES PROGRAMM!

Wir verwenden die Programmierumgebung „Arduino“, welche du für Windows und Mac kostenlos vom Internet runterladen kannst.

Die Abbildung auf der gegenüberliegenden Seite zeigt, wie sich die Arduino-Umgebung präsentiert:

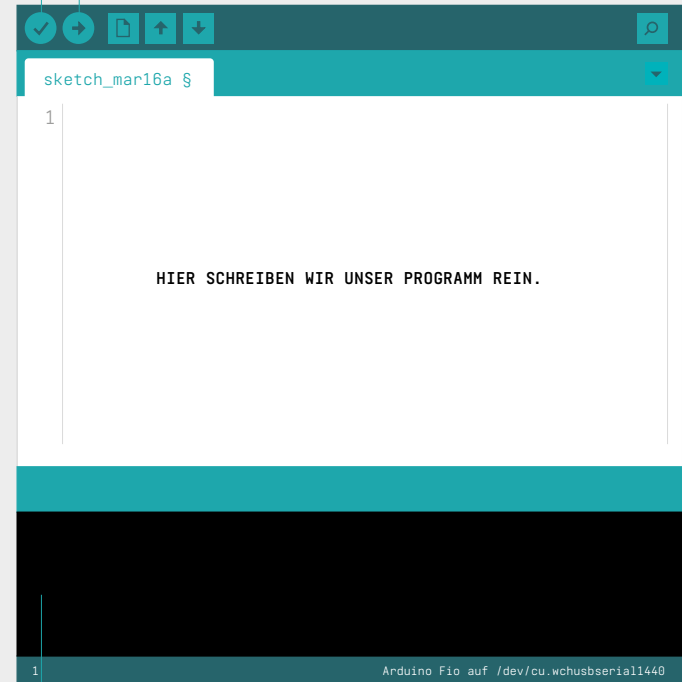
INSTALLATION

Für die Installation haben wir dir eine Anleitung zusammengestellt, diese findest du auf:
www.oxocard.ch/installation



ÜBERPRÜFEN / KOMPILIEREN

HOCHLADEN



HIER ERSCHEINEN FEHLER- UND STATUSMELDUNGEN

Mit dem Texteditor schreiben wir den Programmcode. Damit der Programmcode besser lesbar ist, werden bestimmte Teile der Sprache farblich hervorgehoben.

Der Übersetzer, englisch „Compiler“, ist ein Programm, das den Programmcode der Programmiersprache in die Maschinensprache übersetzt. Damit dies gelingt, müssen wir in allem ganz präzise sein. Wenn wir Fehler machen, hilft uns der Compiler in der Regel mit Tipps. Du startest den Compiler, indem du auf „Überprüfen/kompilieren“ drückst.


Damit der Code vom Computer auf die OXOcard übertragen werden kann, ist eine Software zum Hochladen des Maschinencodes erforderlich. Man nennt sie auch Brennsoftware. Man „brennt“ damit quasi den Maschinencode in den Chip ein. Und das ist nachhaltig: du kannst sogar den Akku entfernen und das Programm bleibt trotzdem erhalten.

Sobald du die Software nach Anleitung installiert hast, starte die Arduino-Umgebung.

ERSTER TEST

1. Verbinde deine OXOcard über USB mit deinem Computer.
2. Öffne aus dem Datei-Menü: Beispiele - OXOcard - Sample 1

Auf dem Bildschirm werden dir jetzt ein paar Zeilen C++-Programmcode angezeigt.



```
sample 1
1 #include "OXOCardRunner.h"
2
3 void setup() {
4   clearDisplay();
5 }
6
7 void loop() {
8
9   clearDisplay();
10  delay(1000);
11
12  turnDisplayOn();
13  delay(1000);
14 }
```

1 Arduino Fio auf /dev/cu.wchusbserial1440

3. Bevor wir erklären, was diese Zeilen genau bedeuten, wähle aus dem Menü „Sketch“ die Funktion „Überprüfen/kompilieren“ und dann die Funktion „Hochladen“.

Nach einer Weile sollten die LEDs auf der Karte blinken.

Dies ist ein erster Funktionstest. Wenn alles geklappt hat, hast du jetzt erfolgreich ein erstes C++-Projekt in Maschinensprache übersetzen lassen und dann auf den Chip der OXOcard gebrannt.

Klappt etwas nicht? Schau dir unsere Hilfestellung unter www.oxocard.ch an.

Wir schauen jetzt den Beispiel-Code ein wenig genauer an. Betrachte den hervorgehobenen Teil und beachte den Rest vorerst nicht:

```
#include „OXOCardRunner.h“

void setup() {
  clearDisplay();
}

void loop() {
  clearDisplay();
  delay(1000);

  turnDisplayOn();
  delay(1000);
}
```

Wir haben vier Zeilen. Du kannst darin englische Wörter erkennen, die etwas ungewöhnlich geschrieben sind. Dies sind Befehle, die wir für dich vorbereitet haben. Im Fachjargon heissen sie Funktionen. Sie teilen der OXOcard mit, was sie zu tun hat. Die drei Funktionen haben folgende Bedeutung:

- `clearDisplay` schaltet alle 64 Pixel aus (`clear display`).
- `delay` lässt die OXOcard ein bisschen warten
- `turnDisplayOn` schaltet alle 64 Pixel an (`turn display on`)

Die Funktion `turnDisplayOn()`; schaltet also die Pixels ein. Die Funktion `delay(1000)`; lässt die OXOcard etwas warten. Die Funktion `clearDisplay()`; schaltet sie wieder aus und dann wird mit `delay(1000)`; wieder etwas gewartet.

AUFGABE

Ersetze beim zweiten „`delay(1000);`“ die Zahl 1000 durch 5000 und starte das Programm erneut („Überprüfen/kompilieren“ und danach „hochladen“).

Wenn alles richtig geklappt hat, sollten die Pixel nun fünf Sekunden leuchten und dann für eine Sekunde ausgeschaltet werden. Experimentiere mit verschiedenen Zahlen zwischen 0 und 10000.



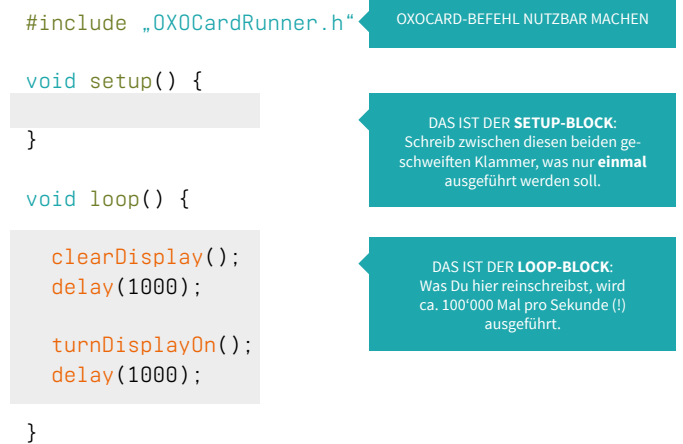
Gratuliere, du hast dein erstes C++-Programm angepasst! Du kannst jetzt den USB-Stecker abziehen und die Karte wird – solange der Strom reicht – so blinken, wie du sie programmiert hast!

Du hast nun vermutlich schon verstanden, dass man bei den Funktionen `turnDisplayOn/clearDisplay` nichts mehr sagen muss, bei `delay` der Computer aber noch wissen muss, wie lange er warten soll. Diese Angabe nennt sich „Parameter“. Die meisten Funktionen brauchen solche Zusatzinfos, sonst ist nicht klar, was zu tun ist.



Das Bild zeigt dir, wie die Funktionen aufgebaut sind. Es beginnt immer mit einem Namen. Da in dieser Programmiersprache Leerzeichen in Namen nicht erlaubt sind, sind die Worte zusammengeschrieben und da `turndisplayon` nicht so gut lesbar ist wie `turnDisplayOn`, hat der Programmierer der Funktionen entschieden, jedes neue Wort mit einem Grossbuchstaben zu beginnen. Die Parameter werden wie Kommentare gleich anschliessend in Klammern gesetzt. Bei `delay` hast du ja schon verstanden, dass man dort eine Zahl angeben muss. Die Zahl teilt dem Computer mit, wie viele Millisekunden er warten muss.

Auf dem folgenden Bild siehst du den Aufbau des Programms.



Funktionen können zu Blöcken zusammengefasst werden. Diese Blöcke sind auch wieder Funktionen. In der Arduino-Umgebung gibt es immer die zwei Funktionen `setup` und `loop` die die Programmiererin oder der Programmierer – sprich du – mit weiteren Anweisungen ausfüllen kann. Hierbei wird alles, was du zwischen die geschweiften Klammern nach `setup()` reinschreibst, ein einziges Mal ausgeführt, wenn der Computer frisch gestartet wird. Anders ist der Block bei `loop()`: alles, was du hier reinschreibst, wird immer wieder ausgeführt. Hier zeigt sich die enorme Geschwindigkeit der OXOcard: Der `loop()` wird ungefähr 100'000 Mal pro Sekunde aufgerufen. Wenn du in `loop()` die Anweisung `delay(1000)` reinschreibst, wartet der Computer eine Sekunde und ruft so `loop()` nur noch einmal pro Sekunde auf. Gecheckt? 😊 Falls nicht, keine Angst. Wir werden noch viel damit zu tun haben und bei jedem Beispiel wird es etwas klarer werden.

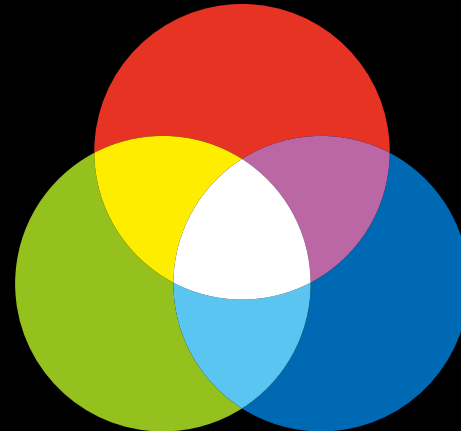
WIE ZEIGT DER COMPUTER BILDER AN?

Wie du vielleicht schon weisst, bestehen Displays aus Bildpunkten, die Pixel genannt werden. Früher konnte man diese noch gut von blossem Auge sehen. Bei neueren Smartphones ist die Pixeldichte so hoch, dass ein einziger Pixel kleiner als ein Zehntel eines Millimeters ist. In einer Fläche von 1x1 Millimeter sind also über 100 Pixel zu finden!

Mehrfarbige Bildschirme

Bei mehrfarbigen Bildschirmen finden wir pro Pixel drei Bildpunkte, einen roten, einen grünen und einen blauen. Wenn man diese Farben zusammenmischt, kann man jede beliebige Farbe kreieren, je nach der Stärke der einzelnen Farben. Du hast vermutlich bereits Erfahrung mit dem Mischen von Farben gemacht. Wenn du Wasserfarben mischst, wird die Farbe immer dunkler, je mehr Farbe du nimmst. Wenn man mit Licht arbeitet, ist es gerade umgekehrt: Je mehr Farben du mischst, desto heller wird die Farbe. Rot, grün und blau zusammen ergeben weiss. Das weisse Licht der Sonne ist also eine Mischung aus allen Farben des Regenbogens.

Auch die OXOcard hat ein Display. Die Pixel sind mit 5mm Seitenlänge deutlich grösser als die auf einem Smartphone. Man findet so grosse Pixel beispielsweise an Leuchtwänden in Stadien. 8 x 8 Pixel sind 64 Bildpunkte, die du einzeln ein/ausblenden kannst. Zudem kannst du die Lichtstärke pro Punkt individuell regeln. Damit hast du eine ganze Menge an Möglichkeiten, die wir nun gemeinsam entdecken wollen.



WIR VERSUCHEN ALS ERSTES, EIN PIXELBILD ZU ZEICHNEN.

AUFGABE

Beschaffe dir ein Häuschenpapier und umrande mit einem Bleistift ein Rechteck mit einer Seitenlänge von je 8 Häuschen, wie im nächsten Bild dargestellt.

Versuche nun, ein Herz zu zeichnen, indem du die entsprechenden Häuschen ausfüllst.

WICHTIG:

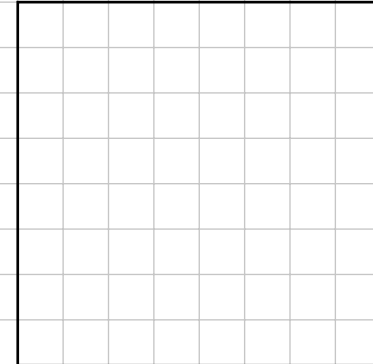
Du hast nur ganze Häuschen zur Verfügung.
Dein Herz wird also etwas eckig, sieht aber auch sehr cool aus 😊.



LÖSUNG - SEITE 162

8 HÄUSCHEN

8 HÄUSCHEN




Wenn du soweit bist, können wir den nächsten Schritt in Angriff nehmen: die Übertragung in computerverständliche Bits. Für unser erstes Experiment verwenden wir ein Bit pro LED, d.h. pro Reihe brauchen wir 8 Bits. Ein Bit kann Null („0“) oder Eins („1“) darstellen. Wir definieren nun, dass eine Eins die LED leuchten lässt und bei Null kein Strom fließt.

Wenn wir unser Herz als Basis nehmen, könnte eine Übertragung in Bits so aussehen:

	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	
	0	0	1	0	1	0	0	0	
	0	1	0	1	0	1	0	0	
	0	1	0	0	0	1	0	0	
	0	0	1	0	1	0	0	0	
	0	0	0	1	0	0	0	0	
	0	0	0	0	0	0	0	0	

Jetzt ist alles vorbereitet und wir können wieder etwas programmieren. Ziel ist es, das Herz auf der LED-Matrix darzustellen. Hierzu musst du lediglich wieder ein vorbereitetes Programm anpassen.

 Öffne hierzu die Datei `sample2`.

Du findest darin eine neue Funktion, die `drewPixels` heisst und acht, mit Komma getrennte Reihen von Nullen und Einsen enthält. Jeder Zeile enthält ein Bit-Folge von 8 aufeinanderfolgenden Bits, die mit „0“ oder „1“ initialisiert sind. Jede Zeile beginnt mit „0b“ - dies teilt dem Computer mit, dass jetzt Bits kommen.

Das Programm zeigt in der Mitte vier Bits auf „1“ gesetzt. Wenn Du es startest, siehst Du ein kleines Rechteck. Mit diesen Pixeln kann man also zeichnen!

AUFGABE

Übertrage die Bits aus deinem Bild in das Programm. Ein Beispiel: ersetze auf Zeile 11 `0b00000000` mit `0b00101000`. Dies entspricht der ersten Pixelzeile unseres Herzchens.

LÖSUNG - SEITEN 164 - 165



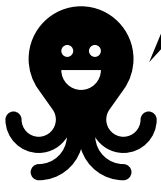
Siehst du das Herz, das wir mit den Einsen und Nullen gezeichnet haben? So ist jede Grafik auf dem Computer aufgebaut.

Starte nun dein Programm mit „Überprüfen/kompilieren“ und danach „hochladen“. Herzliche Gratulation! Du hast soeben ein erstes Bild auf einen Computerbildschirm gezeichnet! Das kann nicht jeder von sich behaupten. 😊

Vielleicht hast du bemerkt dass man beim Programmieren sehr präzis sein muss. Der C++-Compiler prüft das Programm, bevor es übersetzt wird und gibt dir eine Fehlermeldung, wenn beispielsweise ein Strichpunkt fehlt oder man einen Befehl falsch geschrieben hat. Manchmal sind die Fehlermeldungen aber auch unverständlich. Unser Tipp: Ändere immer nur etwas Kleines und führe regelmässig „Überprüfen/kompilieren“ aus.

SO GEHT ES SCHNELLER

Um „Überprüfen/kompilieren“ auszuführen, kannst du auch das Tastaturkürzel Ctrl+R bzw. CMD+R auf dem Mac verwenden.



WEITERE EXPERIMENTE

Übung macht den Meister! Deshalb haben wir dir hier ein paar weitere Ideen für eigene Experimente zusammengestellt:

BLINKEN

Lass dein Herz blinken. Dies ist eine Kombination von Beispielen, die Du bereits kennst. Geh vom letzten Beispiel aus und füge `clearDisplay()` und `delay(1000)` ein.

PLAY

Zeichne auf Karo-Papier zwei Bilder und lasse diese nacheinander abspielen. Du kannst den Befehl `displayImage` so oft verwenden, wie Du möchtest.

BUCHSTABEN

Versuche, die Buchstaben des Alphabets mittels 8x8-Pixelbilder darzustellen. Kann man alle Buchstaben mit so wenig Pixel umsetzen?

WIR BASTELN UNS EINE SMILEY-MASCHINE!

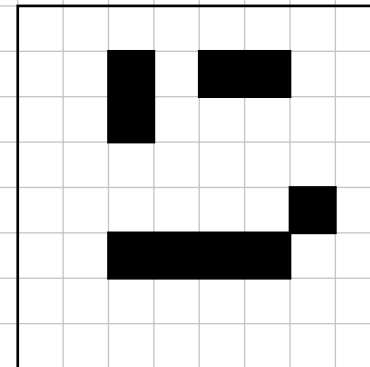
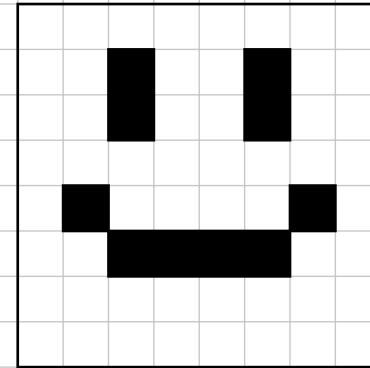
In diesem Kapitel werden wir aus der OXOcard ein Gerät machen, das auf Buttonklicks mit Smileys reagieren kann. Du hast ja bereits Erfahrung damit gemacht, wie man Bilder macht. Für unsere Zwecke brauchen wir zwei Smileys, einen lachenden und einen zwinkernden.

AUFGABE

Erzeuge aus den beiden Smileys je ein Pixelbild. Du weißt ja schon, wie das geht. Ändere dann das Programm aus dem vorigen Kapitel so ab, dass die beiden Smileys abwechselnd im Sekundentakt angezeigt werden.

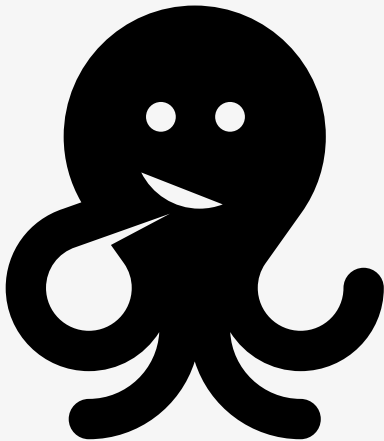


LÖSUNG - SEITE 166



WEITERE EXPERIMENTE

Wenn du möchtest, kannst du natürlich auch noch weitere Pixelbilder erzeugen und diese hintereinander abspielen lassen. So lassen sich sogar kleine Animationen und Filmchen machen! Deiner Kreativität sind hier keine Grenzen gesetzt.



Wir möchten nun einen grossen Schritt weitergehen und dir bedingte Anweisungen zeigen. Damit lassen sich Zustände z.B. Werte von Sensoren oder eben Buttonklicks abfragen und wir können das Programm so anpassen, dass es diese Informationen berücksichtigt.

Unser Beispiel mit den Smileys soll so funktionieren:

Unter der LED-Matrix findest du auf der OXOcard drei Buttons. Wenn du links drückst, soll das Programm einen lachenden Smiley anzeigen und wenn du den Button ganz rechts drückst einen zwinkernden Smiley.

Etwas präziser beschrieben: Wenn der linke Button gedrückt ist, löschen wir zuerst alle Pixel, zeigen dann als zweites einen lachenden Smiley an und warten dann als drittes eine Sekunde.

In C++ „übersetzt“ schreiben wir folgendes:

„if“ ENGLISCH FÜR „WENN“

DIESER BEFEHL CHECKT OB DER BUTTON GEDRÜCKT IST

```
if (isLeftButtonPressed() ) {  
    displayImage( 0b00000000,  
                 0b00100100,  
                 0b00100100,  
                 0b00000000,  
                 0b01000010,  
                 0b00111100,  
                 0b00000000,  
                 0b00000000);  
    delay(1000);  
}
```

DAS IST WIEDER EIN BLOCK AUS GESCHWEIFTEN KLAMMERN. DIE BEFEHLE WERDEN NUR AUSGEFÜHRT, WENN DER BUTTON GEDRÜCKT IST.

Du findest ein Muster, das wie folgt aussieht:

```
if (...) {  
...  
}
```

`if` ist englisch und bedeutet „wenn“. Wenn die Anweisung in der runden Klammer eintritt, führe die Befehle in den geschweiften Klammern aus. Mit den verschiedenen Klammern können wir dem Computer mitteilen, was zusammen gehört.

Du kannst folgende drei Blöcke brauchen:

```
if (isLeftButtonPressed() ) {
```

```
}
```

FÜHRT DIE BEFEHLE AUS,
WENN LINKS GEDRÜCKT
WIRD.

```
if (isMiddleButtonPressed() ) {
```

```
}
```

FÜHRT DIE BEFEHLE
AUS, WENN IN DER MITTE
GEDRÜCKT WIRD.

```
if (isRightButtonPressed() ) {
```

```
}
```

FÜHRT DIE BEFEHLE
AUS, WENN RECHTSS
GEDRÜCKT WIRD.

AUFGABE

Schreibe nun ein Programm, das den lachenden Smiley anzeigt. Wenn du auf den rechten Knopf drückst, löschst du den lachenden Smiley, zeigst den zwinkernden an und wartest eine Sekunde.

Ist dir diese Aufgabe zu schwierig?
Kein Problem. Du findest das Programm unter der Bezeichnung:



„sample5“ im Beispiel-Ordner.

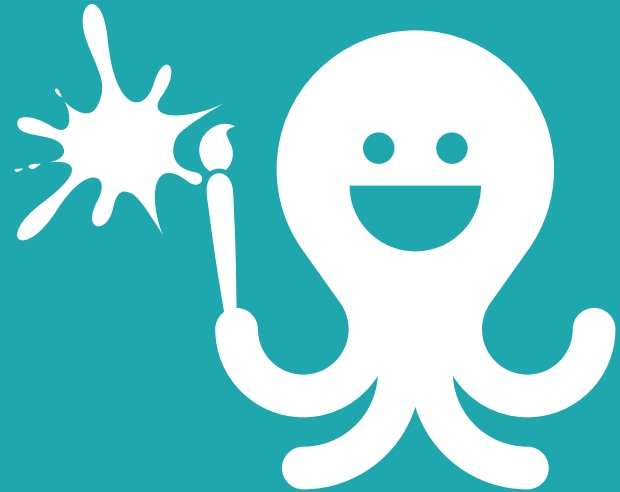
Ein Tipp:

wenn das Programm nicht funktionieren sollte oder du zu wenig Knobelspass an der Sache hast, tippe das Beispiel ab. Dabei lernst du auch eine ganze Menge.

LÖSUNG - SEITE 167



ZEICHNEN MIT GRAFIK-BEFEHLEN



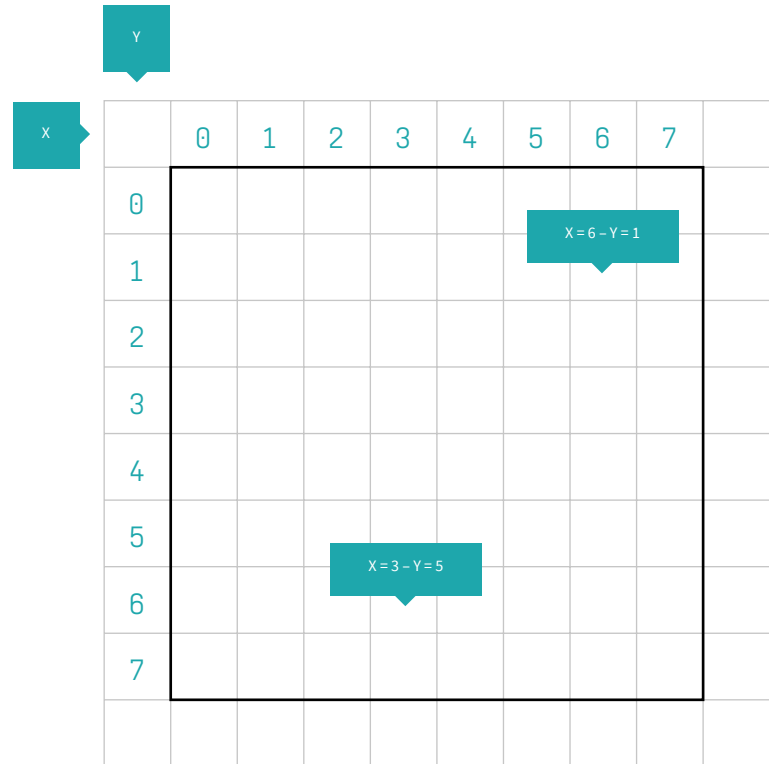
X- UND Y-KOORDINATEN

Du hast jetzt gesehen, wie man auf einem Computer eine Pixelgrafik macht und auch schon erfahren, dass dies relativ kompliziert sein kann, obwohl wir nur 64 LED's haben. Bei einem FullHD-TV haben wir 1920 auf 1080 Pixel und davon jeweils 3 für rot, grün und blau, insgesamt also 6.2. Mio. Pixel! In dieser Auflösung ein Bild von Hand zu zeichnen ist unmöglich. Zum Glück gibt es leistungsfähigere Befehle dafür, die wir nun anschauen wollen.

Wenn wir uns unseren Bildschirm wieder als Karomuster vor Augen führen, haben wir ja jeweils 8 Zeilen zu 8 Spalten. Hast du schon mal „Schiffe versenken“ gespielt? Dort bezeichnet man die Zeilen und Spalten mit Zahlen und Buchstaben, damit man dem Gegenspieler mitteilen kann, wo wir unsere Kanone hinfeuern wollen. Beispielsweise nach B3 oder E5. In der Computergrafik gibt es auch so ein Koordinatensystem mit der X- und der Y-Koordinate. „X“ ist jeweils die Zahl, die die Zellen von links nach rechts bezeichnet, „Y“ von unten nach oben. Anmerkung: im Mathe-Unterricht arbeiten wir ebenfalls mit Koordinatensystemen mit X- und Y-Achse, wobei Y von unten nach oben wächst, d.h. der Nullpunkt ist unten links. Wo sich dieser befindet, ist eine reine Definitionssache. Beim Computer ist es meistens so, wie wir es hier beschreiben und nutzen wollen.

$x=6$ und $y=1$ ist also die Zelle in der 7. Spalte und in der zweiten Zeile. Die Zahlen sind um eins verschoben, weil wir mit 0 anfangen.

Mit diesem Wissen kannst du nun Grafikbefehle verwenden. Es gibt Befehle zum Zeichnen von Punkten, Linien, Rechtecken, Kreisen und Dreiecken, die wir dir nun erklären möchten.



GRAFIKBEFEHLE BRINGEN'S

Du weisst ja schon, dass Befehle immer nach demselben Muster aufgebaut sind. Nach dem Namen kommen runde Klammern und dann ein Strichpunkt. Bei den neuen Zeichnungsbefehlen muss man nun zusätzliche Angaben machen, die wir in die Klammer schreiben und mit Komma trennen. Du kennst das ja schon von der Delay-Funktion, bei der angegeben werden musste, wie viele Millisekunden unser Programm warten soll.

Der einfachste Zeichnungsbefehl, den man zum Zeichnen eines Pixels braucht, heisst `drawPixel` und braucht zwei Parameter, nämlich die X- und die Y-Koordinate des Punkts, der gezeichnet werden soll.

Hierzu ein Beispiel:

```
drawPixel(4,3);
```

Dieser Befehl lässt die LED bei der Koordinate $x=4$ und $y=3$ leuchten. D.h. Spalte 5 und Zeile 4, da wir ja bei Null beginnen und wir daher eins immer im Kopf dazuzählen müssen.

Mit der zweiten Funktion `drawCircle(x,y,r)` kannst du Kreise zeichnen. Beispielweise zeichnet

```
drawCircle(2,3,1);
```

einen Kreis mit dem Mittelpunkt bei den Koordinaten $x=2$ und $y=3$ mit einem Radius von 1.

Bei x/y schreibst du also die Koordinate rein, bei „r“ den Radius, den wir hier zusätzlich brauchen, da sonst der Computer nicht weiss, wie gross der Kreis sein soll.

Mit `drawRectangle(x,y,b,h)` kannst du einfach Rechtecke zeichnen. Auch hier ein Beispiel:

```
drawRectangle(0,0,8,8);
```

zeichnet ein Rechteck am Rand. Die ersten beiden x/y -Parameter kennst du ja schon. „b“ steht für Breite in Pixel und „h“ für Höhe in Pixel. Wir zeichnen also ein Rechteck von links oben bis rechts unten.

Die nächste Funktion zeichnet eine Linie und braucht dafür vier Angaben: die Anfangskoodinate $x1,y1$ und die Endkoordinate $x2,y2$ –`drawLine(x1,y1,x2,y2)` in Action:

```
drawLine(0,0,7,0);
```

zeichnet eine Linie von 0,0, d.h. von oben links, bis 7,0, d.h. oben rechts. Alles klar? ☺

Es gibt auch eine Funktion, die Dreiecke zeichnet. Du kannst damit vielleicht mal Häuser oder sonst was zeichnen, das dir in den Sinn kommt. Die Funktion heisst `drawTriangle(x1,y1,x2,y2,x3,y3)`. Damit der Computer ein Dreieck zeichnen kann, braucht er 6 Angaben, d.h. die drei Koordinaten der Ecken, die das Dreieck aufspannen. Das geht so:

```
drawTriangle(0,0,5,0,3,3);
```

Diese Funktion zeichnet ein Dreieck. Der erste Punkt ist bei 0,0, der zweite bei 5,0, der dritte bei 3,3. Wenn du dir das nicht vorstellen kannst, gibt's nur eins: Üben. 😊 Daher gibt's jetzt wieder eine Aufgabe:

AUFGABE

Du hast jetzt gesehen, wie man mit Bits Pixelbilder machen kann. Versuche nun ein Gesicht mit den neu erworbenen Grafikbefehlen zu zeichnen. Damit's einfacher geht, haben wir dir ein Beispiel zusammengestellt, das du als Basis nutzen kannst.



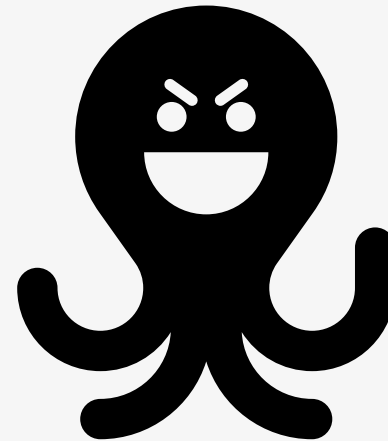
Du findest es unter dem Namen „sample6“



LÖSUNG - SEITE 168

WOOOOW!

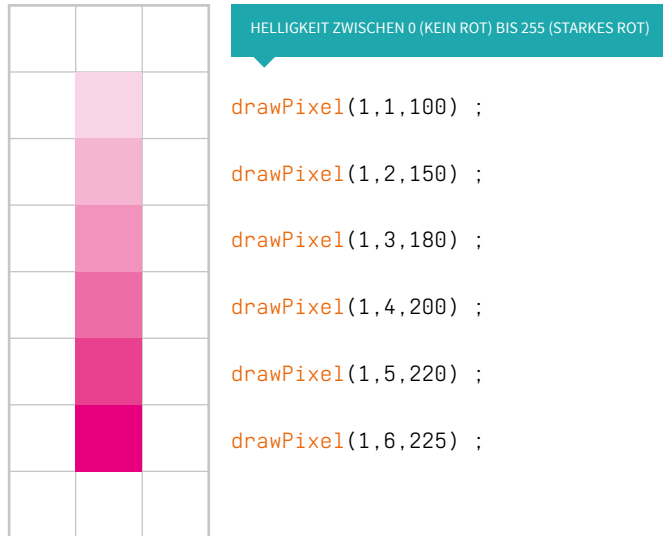
Wenn du bis hier durchgehalten hast, gratulieren wir dir ganz herzlich! Du kannst bereits Bilder darstellen, Animationen machen und diese mit den Buttons verknüpfen. Macht's Spass?



DARF ES ETWAS WENIGER HELL SEIN?

Bis jetzt haben wir die nur LED's ein- und ausgeschaltet, aber unsere LED's sind nicht binär, das heisst man kann damit verschieden helle Rottöne darstellen. Das coole dabei ist, dass es genau die gleichen Befehle sind, die du bereits kennst. Um die Helligkeit anzugeben, braucht es einfach einen weiteren Parameter.

Das folgende Beispiel zeigt, wie man den neuen Wert, den man in englisch „brightness“ (Helligkeit) nennt, nutzen kann:



Die Helligkeit kann man mit einem Wert zwischen 0 und 255 einstellen. Wenn man – wie bisher – bei den Zeichnungsbefehlen nichts angibt, nimmt der Computer an, dass du maximale Helligkeit möchtest, d.h. 255. Die Helligkeit lässt sich bis 0 runterdimmen, wobei „0“, d.h. gar kein rot, die LED auslöscht. Oder anders ausgedrückt: Damit kannst du Dinge auf Bildern löschen. 😊

Wir haben wieder ein Beispiel für dich vorbereitet, bei dem du mit der Helligkeit spielen kannst.



Starte das Beispiel [sample6b](#).

Wir haben dort ein wenig mit der Helligkeit rumgespielt und lassen die ganze Grafik im Sekundentakt blinken.

AUFGABE

Aktuell zeichnen wir die ganze Grafik und löschen diese dann auch wieder. Deine Aufgabe ist es nun, nur das Rechteck und das einzelne Pixel zu löschen. Knifflig? Wie war das nochmal mit der Helligkeit, die auf „0“ gestellt wird? Achtung: Knobelalarm!

LÖSUNG - SEITEN 170 - 171



LET'S
MOVE!



RECHNEN

Nebst dem Darstellen von Bildern hat der Computer natürlich noch eine andere sehr wichtige Fähigkeit: das Rechnen. Zu diesem Zweck brauchen wir einen neuen Befehl. Der Befehl heisst `drawNumber` und zeichnet mit unseren Pixeln die Zahlen von 0 bis 99. Der Computer kann natürlich mit viel grösseren Zahlen rechnen, jedoch können wir diese mit unseren Pixeln nicht darstellen.

Das Rechnen ist natürlich eine der Königsdisziplinen des Computers und auch der Grund, warum man solche überhaupt entwickelt hat. Auch unser kleiner Computer ist fähig, beliebige mathematische Funktionen, wie Quadratzahlen oder Wurzeln zu berechnen. Vielleicht hast du auch schon mal was vom Logarithmus oder der Sinusfunktion gehört, die man ebenfalls berechnen kann. Falls nicht, spielt es auch keine Rolle, denn wir konzentrieren uns hier auf die Grundrechenoperationen, Addition „+“, Subtraktion „-“, Multiplikation „*“ und Division „/“. Statt dem Multiplikationspunkt brauchen wir auf dem Computer den Stern „*“, statt dem Doppelpunkt als Divisionszeichen, brauchen wir den Querstrich „/“.

AUFGABE



Starte das Programm `sample7`.

Du findest darin den neuen Befehl „`drawNumber`“ und dahinter eine Rechenaufgabe als Parameter. Starte das Programm und checke, was der Computer auf dem Screen darstellt. Berechne den Inhalt der Klammer im Kopf und überprüfe, was der Computer berechnet hat. Stimmt das Resultat?

LÖSUNG - SEITE 172

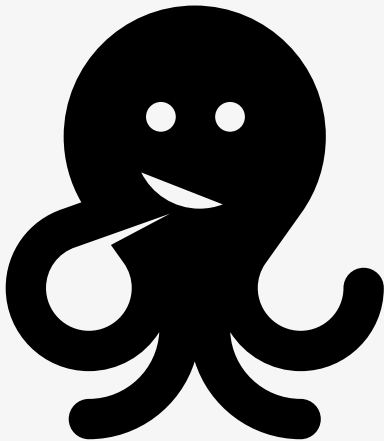


ZUSATZAUFGABE

Ändere die Rechnung nun nach deinem eigenen Geschmack ab. Du kannst sehr lange Reihen machen, wobei der Computer zum Ausrechnen dafür nur einige wenige Millisekunden Zeit brauchen wird.

WICHTIG ZU WISSEN

Wir haben die Zahlen bewusst beschränkt, daher bleibe im Rahmen der Zahlen von 0 bis 99, da grössere Zahlen nicht dargestellt werden können. Aktuell kann `drawNumber` auch nicht mit negativen Zahlen umgehen.



VARIABLEN UND WEITERE IF-ANWEISUNGEN

Du hast jetzt gesehen, dass du mit dem Computer auch rechnen kannst. Du kannst dies überall machen, wo eine Zahl angegeben wird, beispielsweise auch bei der bereits bekannten `Delay`-Funktion, bei der wir ja Millisekunden angeben müssen, damit wir den Programmablauf für eine Weile unterbrechen. Du kannst also

```
delay(5000);
```

oder

```
delay(5 * 1000);
```

schreiben, wenn das für dich lesbarer ist.

Wir möchten jetzt ein weiteres neues Konzept einführen, mit dem spannende Dinge programmiert werden können. Es geht um Variablen. Eine Variable ist ein Platzhalter für einen Wert. Wir beginnen ganz einfach und schreiben folgendes:

```
int eineSekunde = 1000;
```

Mit dieser Anweisung weist du der Variablen „`eineSekunde`“ den Wert 1000 zu. Du kannst jetzt mit der Variablen rechnen, als wäre es eine Zahl, wie hier dargestellt:

```
int eineSekunde = 1000;  
delay(5 * eineSekunde);
```

WIR BRAUCHEN EINE GROSSE ZAHL

SPEICHERE DIESEN WERT

```
int eineSekunde = 1000 ;
```

BELIEBIGER NAME

Du kannst dir Variablen wie angeschriebene Schubladen vorstellen. In der Schublade mit der Anschrift „eineSekunde“ steht die Zahl 1000;

Was bedeutet das `int`? Wenn wir eine Variable definieren, müssen wir dem Computer mitteilen, wie viel Platz er dafür reservieren soll. Bildlich gesprochen, kannst du dir das vorstellen, wie ein Papier. Je grösser das Papier ist, desto mehr kannst du draufschreiben.

Wenn du folgendes definierst:

```
byte eineVariable = 57;
```

teilst du dem Computer mit: „Bitte eine Schublade mit der Aufschrift „eineVariable“ reservieren für eine Zahl zwischen 0 und 255. Grössere Zahlen haben in der Schublade nicht Platz und werden abgeschnitten. Wenn du eine grössere Zahl zuweist, teilt der Computer diese Zahl durch 255 und weist der Variablen nur den Rest zu.

Wenn du eine grössere Zahl speichern möchtest, müssen wir mehr Platz schaffen. `int` ist die nächstgrössere Schublade, die Zahlen zwischen -32'768 und 32'768 speichern kann. Ideal also für unsere Zahl 1000. 😊

Nun geht's an die nächste Aufgabe.

```
#include „OX0CardRunner.h“
void setup() {
  clearDisplay();
}
byte a;
byte b;

void loop() {
  a = 5;
  b = 10;
  drawNumber(5+a*b/2);
}
```

Es werden zwei Variablen, also zwei Schubladen mit der Beschriftung a und b definiert. Und dann rechnen wir damit was aus und stellen den Wert dar.

AUFGABE



Bitte lade das Beispiel `sample7b`.

Halt! Bevor du das Programm startest eine Frage. Welche Zahl wird dargestellt? 😊

LÖSUNG - SEITE 173



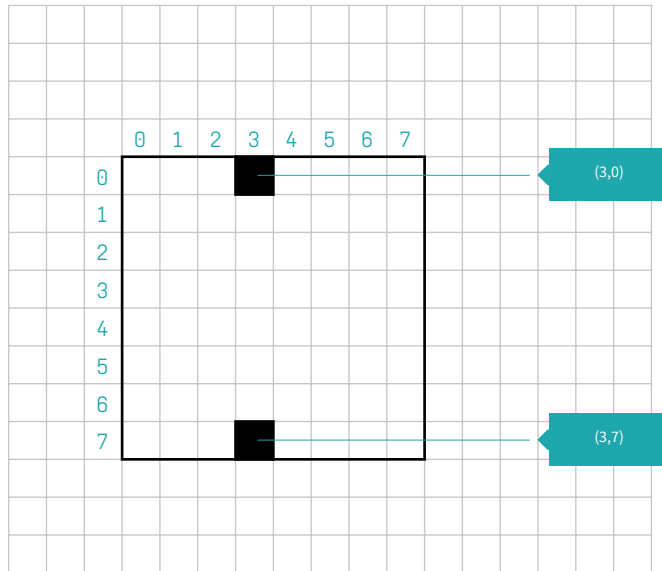
JETZT ANIMIEREN WAS

Mit Variablen kann man natürlich viel mehr machen, als nur rechnen. Eine Variable kann man zum Beispiel als Zähler brauchen und mit einem Zähler kann man Dinge animieren.

Folgendes Beispiel:

Stell dir vor, du möchtest ein Spiel programmieren, bei dem Pixel vom oberen Rand des Bildschirms nach unten fallen. Nehmen wir mal einen Pixel, der ab der Position $x=3$, $y=0$ startet und bei $x=3$ und $y=7$ endet.

Hier eine grafische Darstellung davon:



Wenn der Pixel runterfallen soll, ändert sich die Y-Koordinate und nimmt alle Werte zwischen 0 und 7 an, wir haben also folgende Reihe: 0,1,2,3,4,5,6,7. Das möchten wir nun mittels einer Variablen programmieren.

Da das Beispiel etwas kompliziert ist, fangen wir zuerst einmal ohne den Pixel, nur mit den Zahlen an, an die Aufgabe heranzugehen. Ziel des ersten Schrittes ist es, ein Programm zu schreiben, dass die Zahlen 0 bis 7 nacheinander darstellt.

AUFGABE



Lade die Datei `sample7c`.

Schau' dir das Programm genau an. Versuche zu erklären, warum die Zahlen zwischen 0 und 7 aufsteigend dargestellt werden.

LÖSUNG - SEITE 174



Schau dir folgende Grafik an:

```
byte a;
```

VARIABLE A IST UNSER ZÄHLER, DEN WIR AUSSERHALB DES LOOP-BLOCKS DEFINIEREN.

```
void loop() {  
  drawNumber(a)  
  delay(500);
```

```
  if (a < 8) {  
    a = a + 1;  
  }
```

WENN A KLEINER ALS 8 IST, ERHÖHEN WIR DEN WERT DER VARIABLEN UM EINS, D.H. BEI JEDEM DURCHGANG WIRD A UM EINS ERHÖHT.

```
  else {  
    a = 0;
```

WENN A NICHT KLEINER ALS 8 IST, - ODER ANDERS AUSGEDRÜCKT: A IST GRÖßER ODER GLEICH ALS 8, DANN SETZEN WIR A AUF 0 UND ALLES BEGINNT WIEDER VON VORNE.

```
}
```

Der Aufbau ist

```
if (hier steht eine Bedingung) {
```

.. wenn die Bedingung erfüllt ist, wird alles ausgeführt, was hier steht.

```
}  
else {
```

else bedeutet, „sonst“: dieser Block wird ausgeführt, wenn obige Bedingung nicht erfüllt ist.

```
}
```

Schau Dir die Programmzeilen nochmals an... gecheckt? Ist doch ganz einfach. 😊 Drum hier gleich eine Aufgabe:

DOPPELAUFGABE



Lade die Datei sample7c.

Passes das Programm so an, dass es bis 50 zählt (nicht bis 7).

Und gleich noch eine Aufgabe, weil es so cool ist. Wir möchten immer in 5-er Schritten hochzählen. Was musst du hier ändern, damit das klappt?

LÖSUNGEN - SEITE 175



ZUSATZINFOS

Du kannst Variablen auf folgende Arten abfragen:

a > b

a ist grösser als b

a < b

a ist kleiner als b

a == b

a ist identisch mit b

a != b

a ist ungleich b

a >= b

a ist grösser oder gleich b

a <= b

a ist kleiner oder gleich b

Mit der `if`-Anweisung kannst du also Weichen stellen und den Computer, je nach Situation, entweder in die eine oder die andere Richtung lenken. Das ist eines der zentralen Elemente der Programmierung, da wir so auf verschiedene Situationen mit unterschiedlichen Handlungen reagieren können. Beispielsweise kann ich bei einem Spiel prüfen, ob sich zwei Figuren berühren und dann einen Ton ausgeben.

Du hast jetzt ein bisschen mit Variablen gespielt und gesehen, wofür wir `if` noch so brauchen können. Jetzt möchten wir unser einleitendes Beispiel der Animation fertigstellen und dich nicht weiter auf die Folter spannen:



Lade das Beispiel [sample7d](#) und starte es.

Versuche zu verstehen, wie das Programm aufgebaut ist.

Die Zeile

```
byte a;
```

definiert eine Variable für eine Zahl von 0 bis 255.

```
drawPixel(3,a);  
delay(50);  
drawPixel(3,a,0);
```

Diese Zeilen zeichnen ein Pixel in der Spalte 4 und der Zeile „a“, dann wird 50 Millisekunden gewartet, dann wird der Pixel wieder gelöscht. Zuvor haben wir bei Delay immer länger gewartet und du hast dich vielleicht schon gefragt, warum man die Wartezeit in Millisekunden angeben muss.

Das menschliche Auge nimmt fließende Bewegungen erst wahr, wenn pro Sekunde ca. 25 Bilder hintereinander angezeigt werden. Wenn wir also 50ms warten, sind das ca. 20 Bilder pro Sekunde, wodurch wir nicht mehr Einzelbilder sondern eben Bewegung wahrnehmen, wenngleich es bei 20 Bildern noch etwas flackert. Mit kurzen Wartezeiten kann man also coole Animationen basteln. 😊

Den nächsten Block kennst du bereits aus dem vorigen Beispiel:

```
if (a < 7) {  
  a = a + 1;  
} else {  
  a = 0;  
}
```

Auf Deutsch: Wenn a kleiner als 7 ist, dann erhöhe a um eins, andernfalls setze die Variable a auf 0.

AUFGABEN



Lade das Beispiel `sample7d` und starte es.

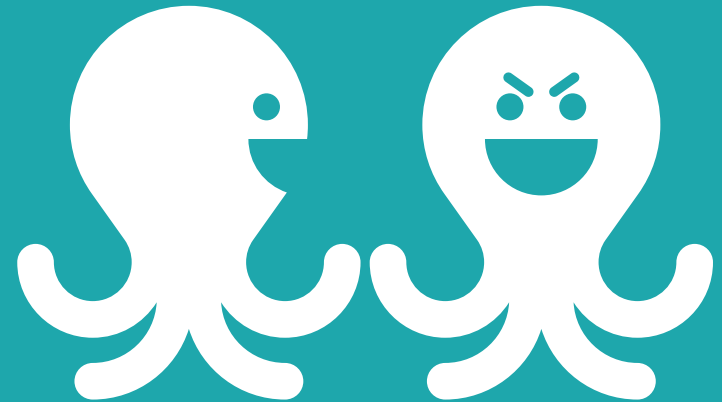
Du kannst jetzt ein wenig experimentieren und folgendes ausprobieren:

- beschleunige oder verlangsame die Animation
- versuche, zwei Pixel gleichzeitig runterfallen zu lassen
- lass die Pixel nur bis zur Mitte fallen.

LÖSUNGEN - SEITE 176



ZUFÄLLIGE BEGEGNUNGEN




DIE ZUFALLSFUNKTION

Wir möchten dir ein weiteres Konzept näherbringen, ohne das fast kein Spiel auskommt: der Zufall. Für uns Menschen ist es das einfachste der Welt, beispielsweise eine beliebige Zahl zwischen 1 und 10 zu nennen oder beim Laufen zufällig die Richtung zu ändern. Für den Computer, der sich immer strikt nach einem Plan verhält, ist Zufall schwierig. Wenn der Computer Sensoren hat, kann er z.B. die aktuelle Temperatur als zufälligen Wert nehmen oder den Neigungswinkel deines Handys. Wenn beides jedoch nicht ändert, wird der Wert immer derselbe sein. Findige Programmierer haben eine Funktion entwickelt, die eine Zufallszahl liefert. Diese können wir ganz einfach nutzen, sie heisst `random`, das englische Wort für zufällig und funktioniert so:

```
byte a = random(7);
```

Du brauchst als erstes wieder eine Variable, die den Zufallswert speichert. Hier haben wir „a“ genommen. Dann gibst du an, wie gross die Zahl höchstens sein soll. In dem Beispiel werden beliebige Werte zwischen 0 und 6 „erfunden“. Jedes Mal, wenn du `random(7)` aufrufst, bekommst du also eine andere Zahl.

Ändere den Parameter der Random-Funktion ab und schau, welche Zahlen präsentiert werden.

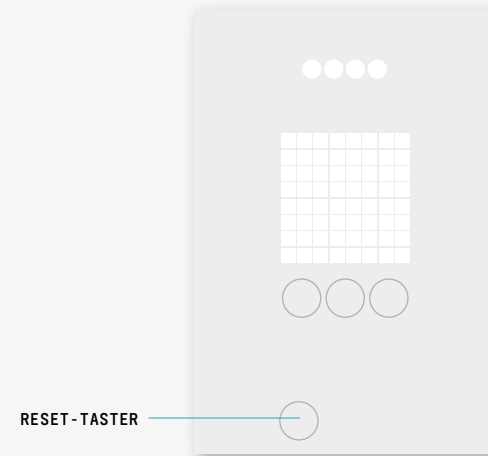
 Öffne hierzu die Datei `sample8`.

Wenn du beispielsweise `random(20)` eingibst, erscheinen nur Zahlen zwischen 0 und 19.

ZUSATZINFOS

Leider sind die Zahlen nicht so zufällig, wie es den Anschein nimmt. Jedes Mal, wenn der Computer neu gestartet wird, fängt „random“ wieder von vorne an und liefert wieder die exakt selben Zahlen.

Du kannst das ganz einfach checken: Klicke auf den Button unten links gleich neben dem USB-Kabel. Dies startet die OXOcard neu.



Merke dir nun die ersten paar Zahlen und dann starte den Computer nochmals. Du siehst: Die Zahlen sind zwar zufällig gewählt, aber es sind immer dieselben.

AUFGABE

Du bist jetzt schon ein Profi, daher werden die Aufgaben nun auch etwas anspruchsvoller.



Auf Basis des Beispiels `sample8` sollst du Folgendes programmieren:


Erstelle einen Kreis mit der Funktion `drawCircle`. Erstelle ein Programm, das zufällig Kreise mit Radien zwischen 0 und 4 zeichnet. Der Mittelpunkt der Kreise soll an den Koordinaten $x=4$ und $y=4$ sein.



LÖSUNG - SEITE 177

ZUFÄLLIGE KUNST

Du kannst deine OXOcard auch zu einem Kunstgenerator programmieren.

 Lade hierzu das Beispiel `sample8c` und starte es.

Sieht das nicht cool aus? Alle LEDS blinken in unterschiedlichen Helltönen und man könnte meinen, die OXOcard erwacht zum Leben. 😊 Dabei haben wir nur drei Zufallszahlen generiert und mit diesen einen Pixel gezeichnet.

Das Programm ist sehr kurz, aber es braucht trotzdem etwas Hirnschmalz, bis man dahinterkommt, was da genau geschieht. Schauen wir uns das im Detail an:

Der folgende Block zeichnet einen einzigen Pixel:

```
byte x = random(8);  
byte y = random(8);  
byte b = random(255);  
drawPixel(x,y,b);
```

Wir wählen zufällig eine X-Position (0 bis 7), zufällig eine Y-Position (0 bis 7) und dann noch eine Helligkeit zwischen 0 und 255 und zeichnen den Pixel. Dann fängt das Programm von vorne an und fährt an anderer Stelle fort. Dadurch, dass die Oxocard so schnell ist, füllt sich die Karte in Sekundenbruchteilen mit 64 zufälligen Pixeln, wodurch diese Animation entsteht.

Wir haben jetzt den grafischen Teil abgeschlossen und dir mit der Random-Funktion zudem eine weitere Möglichkeit für eigene Experimente eröffnet. Fassen wir einmal kurz zusammen, was du alles schon gelernt hast: Du kannst Bilder zeichnen, Pixel verschieben, abfragen, ob jemand einen Button gedrückt hat. Du kannst mit der `if`-Anweisung Weichen stellen und mit der Random-Funktion zufällige Werte generieren. Eigentlich genügen diese Teile bereits, um ein eigenes Spiel zu programmieren. Du wirst vielleicht entgegnen, dass etwas Wesentliches noch fehlt, die Erzeugung von Geräuschen und Tönen. Du hast natürlich vollkommen recht, daher schauen wir uns das gleich als nächstes an.

JETZT
WIRD'S
LAUT

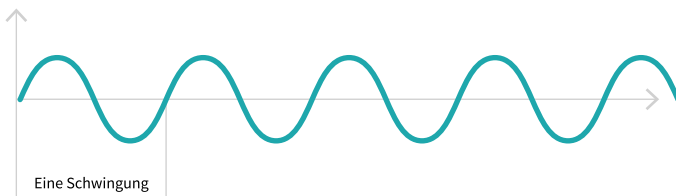


WIE TÖNE ENTSTEHEN

Glücklicherweise haben wir einen kleinen Tongenerator in die Oxocard eingebaut. Mit einem kleinen Lautsprecher, der sich oben rechts in der Karte befindet, kannst du eine ziemlich eindruckliche Lautstärke erzeugen, daher empfehlen wir dir, die folgenden Experimente nur während des Tages und keinesfalls nach zehn Uhr abends durchzuführen. 😊

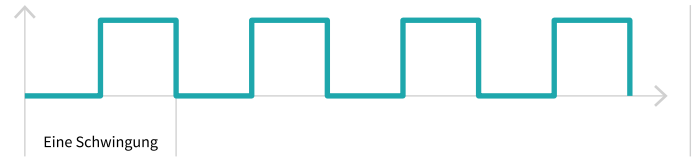
Ein Ton ist eine Schwingung von Luft. Je schneller diese schwingt, desto höher der Ton. Der Schwingung sagt man auch Frequenz. Die Einheit nennt sich Hertz und ist nach dem Physiker Heinrich Hertz benannt – hat also nichts mit dem Herzschlag zu tun.

Das folgende Bild zeigt, wie so eine Schwingung theoretisch aussieht. Bei der Sinus-Schwingung nimmt der Pegel langsam zu und langsam wieder ab. Sie erzeugt einen weichen, aber eher langweiligen Ton:



Die Höhe der Kurve nennt man Amplitude, sie ist verantwortlich für die Lautstärke. Je höher, d.h. je stärker der Ausschlag, desto mehr Luft wird bewegt und desto lauter wird der Ton, bzw. das Geräusch.

Die Oxocard erzeugt den Ton mit einem Rechteck-Generator, der so ein Signal liefert:



Dieses Signal ist etwas krächzender, weniger weich und entspricht diesem typischen Klang, den man aus alten Videogames kennt. Wir Entwickler bei OXON lieben diesen Klang 😊 und hoffen, dass wir unsere Begeisterung dafür auch an dich weitergeben können.

Dass eine Schwingung einen Ton erzeugt, ist schwierig zu verstehen, wenn man nicht selber einmal etwas damit rumexperimentiert hat. Daher wollen wir gleich mit einem Beispiel starten. Damit ein Ton erzeugt werden kann, brauchen wir einen Generator, der in kurzen Abständen eine Spannung an- und ausschaltet. Der Befehl, den wir hierzu brauchen, heisst `tone()` und braucht einen oder zwei Parameter:

```
tone(440);
```

Dieser Aufruf erzeugt einen Ton mit der Frequenz von 440 Hertz. Das ist die Frequenz des sogenannten Kammertons, den man auch eingestrichenes A nennt (a'). Beim Kammer-ton haben sich verschiedene Staaten auf einen Ton geeinigt, damit die Musik, die du nach Noten spielst, überall gleich klingt.



Lade nun das [Beispiel 9](#) und spiele es einmal ab.

Du hörst einen Ton mit der Frequenz von 440 Hertz, dann wartet das Programm eine Sekunde und dann wird mit dem Befehl `noTone()`. Der Ton wieder abgestellt.

AUFGABE

↑ Öffne Beispiel 9

Du kannst mit dem Programm experimentieren. Verändere die Frequenz. Bis wie tief, bzw. wie hoch kannst du den Ton noch wahrnehmen?



LÖSUNG - SEITE 178

DIE SIRENENSCHLEIFE

Im nächsten Beispiel lernen wir ein weiteres wichtiges Element unserer Programmiersprache. Wir haben ja schon gesehen, wie man Zahlen hoch- und runterzählen kann und damit sogar schon Pixel verschoben. Im nächsten Schritt lernst du nun, wie man dasselbe mit einer Schleife machen kann. Wir starten wieder mit einem Beispiel, das du laden und ausführen kannst.

↑ Lade nun das [Beispiel 9b](#).

Wenn du das Programm startest, hörst du eine Tonfolge, bei der der Ton zuerst rasch höher wird und dann wieder zurück auf einen sehr tiefen Ton fällt. Den Befehl für einen Ton, kennst du ja bereits und dass man überall auch eine Variable angeben kann, wann, wo eine Zahl erwartet wird, ist ja auch schon kalter Kaffee. Schauen wir uns mal den ersten Teil des Codes genauer an:

```
for (int i = 0; i < 4000 ; i++) {  
    tone(i);  
}
```

Wir haben hier eine sogenannte `for`-Schleife. `for` steht für „für“ und wir lesen die erste Zeile des Codes wie folgt:

Für eine **Zahl `i` mit dem Anfangswert `0`** durchlaufen wir eine Schleife so lange, wie **`i` kleiner als `4000`** ist, wobei wir bei jedem Durchlauf **`i` um eins erhöhen**.

`for` hat also drei Angaben, die man festlegen muss. Zuerst wird der Anfangswert angegeben, dann die Abbruchbedingung und dann noch, wie sich der Schleifenwert bei jedem Durchgang ändern soll. Hier nochmals in einer übersichtlicheren Darstellung:

UNSERE VARIABLE HEISST `i` UND STARTET BEI 0...

...UND LÄUFT, SOLANGE `i` KLEINER ALS 10 IST, D.H. BIS 9...

...WOBEI WIR BEI JEDEM DURCHLAUF `i` UM EINS ERHÖHEN.

```
for (int i = 0; i<10;i++) {  
      
}
```

DIESER BLOCK WIRD 9x AUFGERUFEN UND NIMMT `i` NIMMT DABEI DIE WERTE 0,1,2,3,4,5,6,7,8 UND 9 AN.

Der Inhalt des Blocks wird also zehnmal durchlaufen, bevor das Programm weitergeht. `i` nimmt während dieser Zeit folgende Werte an:

```
i = 0  
i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
i = 6  
i = 7  
i = 8  
i = 9
```

Was bedeutet das `i++`? Das ist eine häufig gebrauchte Kurzschreibweise für `i=i+1` und bedeutet, dass wir die Variable `i` um eins erhöhen. Du kannst neben `i++` auch `i--` schreiben, dann wird die Zahl in der Variable um eins verringert.

AUFGABE

Man lernt am meisten, wenn man mit den Programmen rumspielt, daher möchten wir dich animieren, eine Sirene zu bauen. Ändere das Beispiel so ab, dass die Sirene kontinuierlich läuft und experimentiere mit der Höhe und der Länge der Töne.



Öffne im Beispiel-Ordner `sample9b`

LÖSUNG - SEITE 179



SPIEL MIR EIN LIED

Es gibt noch einen weiteren coolen Befehl, den wir für dich vorbereitet haben. Mit `playMelody` kannst du ganze Musikstücke abspielen lassen. Der Aufbau ist etwas komplizierter, aber du bist mittlerweile ja schon routiniert. Wir starten mit einem Beispiel.



Lade hierzu bitte [sample 9c](#) und spiele es ab.

Damit wir eine Melodie abspielen können, müssen wir dem Computer mitteilen, welche Töne abgespielt werden sollen und wie lange diese sind. Bis jetzt haben wir bei den Befehlen immer eine fixe Liste von Werten übergeben, aber bei einer Melodie, die beliebig lang ist, kommen wir damit natürlich nicht weiter. Schauen wir uns mal die folgende Zeile an:

```
int melody[] = { NOTE_C5, NOTE_G4, NOTE_G4,
                NOTE_A4, NOTE_G4, 0, NOTE_B4,
                NOTE_C5 };
```

Offensichtlich ist das eine Variable `melody`, die wir hier deklarieren. Sie hat den Typ `int`, d.h. kann eine Zahl zwischen -32768 bis +32767 speichern. Neu ist, dass wir nach dem Namen zwei eckige Klammern angegeben haben. Mit den „[]“ hinter dem Namen teilen wir dem Computer mit, dass wir eine sehr grosse Kiste brauchen, da wir viele `int`-Variablen speichern wollen. Mit den geschweiften Klammern zählen wir danach auf, welche Werte das sind. Hier nochmals als Diagramm:

DIE VARIABLE SPEICHERT
INT-WERTE...

...UND HEISST MELODY.

DAMIT SAGEN WIR: WIR MÖCHTEN
MEHRERE WERTE ANGEBEN.

```
int melody[] = { NOTE_C5, NOTE_G4, NOTE_G4,
                NOTE_A4, NOTE_G4, 0, NOTE_B4,
                NOTE_C5 };
```

ALLE WERTE GIBST DU IN {} KLAMMERN
UND MIT KOMMA GETRENNT AN.

Du kannst eine beliebige Anzahl Werte hinten aufführen, beispielsweise nur drei oder einhundert. Alle Werte müssen innerhalb der geschweiften Klammern stehen und mit Komma getrennt werden.

Bevor wir eine Melodie spielen können, müssen wir dem Computer auch noch mitteilen, wie lange die Töne dauern. Nach demselben Prinzip definieren wir eine weitere Variable:

```
int durations[] = { 250, 125, 125, 250, 250, 250,
                   250, 250 };
```

Darin definieren wird wieder in Millisekunden, wie lange der jeweilige Ton dauern soll.

Als letztes führen wir die Befehl `playMelody` aus:

```
playMelody(melody, durations, 8);
```

Dieser Befehl braucht drei Parameter, die Liste der Töne `melody`, die Liste der Tondauer `durations` und als letztes noch die Anzahl der Töne, die abgespielt werden sollen – hier 8.

AUFGABE

Du kannst dir vorstellen, was jetzt kommt. 😊
Du darfst selber etwas komponieren!!
Versuch dich zuerst an einem einfachen Beispiel, der Tonleiter. Falls du möchtest, kannst du auch mutiger werden und beispielsweise schauen, ob du auf dem Internet Noten eines Musikstücks findest. Wandle die Noten in die Konstanten der unserer Tabelle in Anhang um und programmieren deinen Song!



LÖSUNG - SEITE 180

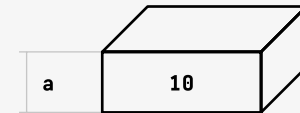
ARRAY - VARIABLEN

Die eckigen und runden Klammern teilen dem Computer mit, dass du eine Variable für mehrere Werte brauchst. Diese nennt man Array-Variablen. Die Variable kannst du dann überall in deinem Code brauchen. Wenn du einen einzelnen Wert lesen möchtest, machst du das so:

```
int zahlen[] = {89,90,91};
```

```
drawNumber(zahlen[0] + zahlen[1] + zahlen[2]);
```

Wir deklarieren in dem Fall einen Array mit drei Zahlen, die wir dann alle zusammenzählen und ausgeben. Anmerkung: Das erste Element im Array hat immer die Zahl „0“!!

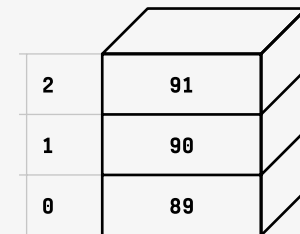


```
int zahlen[] = {89,90,91};
```

= zahlen[2]

= zahlen[1]

= zahlen[0]




DAS SPIEL MIT DER BESCHLEUNIGUNG



WO IST OBEN?

Die OXOcard verfügt über einen Beschleunigungssensor, wie er heute in fast jedem Handy und Tablet eingebaut ist. Damit kann ein Gerät Bewegungen oder Erschütterungen erfassen und mit dieser Information etwas anstellen, beispielsweise ein Spiel steuern oder eine App bedienen. Du hast damit sicherlich schon gespielt und bist schon ein Profi in der Nutzung. Jetzt erfährst du, wie's funktioniert.

Bei unserem ersten Experiment nutzen wir einen Befehl, der `getOrientation` heisst und dir eine Zahl zwischen 1 und 4 zurückgibt, je nachdem, wie die Karte positioniert ist. Der Befehl nutzt den Beschleunigungssensor, der um drei Achsen misst, ob sich die Karte bewegt.

 Starte das Beispiel `sample10`. Wenn du die Karte in verschiedene Positionen legst, erhältst du die Zahlen 1 bis 4 angezeigt. Findest du alle vier?

Wenn die Karte flach auf dem Tisch liegt, wird 1 ausgegeben. Wenn du die Karte hochkant hochhältst, entspricht das dem Wert 4, quer ist es die 3. Und wenn du die Karte umgekehrt auf den Tisch legst, ist es die 2.

Mit dieser Information kannst du schon einiges programmieren. Du kannst beispielsweise unterscheiden zwischen liegend (1) und aufrecht (4) und kannst so je nach Position einen anderen Smiley anzeigen. Eigentlich schreit das ja nach einer Aufgabe! 😊

AUFGABE

Wenn die Karte liegt, schläft der Smiley. Wenn du die Karte aufrecht hältst, lacht er dich an! Nutze `getOrientation` und den bereits bekannten Befehl „`drawImage`“ für diese Aufgabe.



Öffne das Beispiel `sample10`.

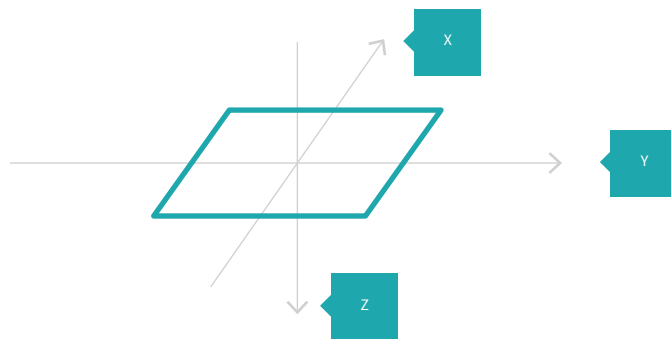
LÖSUNG - SEITE 181



GRAVITATION

Der Beschleunigungssensor misst Beschleunigen und Abbremsen. Wenn du die Karte ruhig hältst, kann der Sensor nichts messen. Doch im vorigen Kapitel haben wir eigentlich nur die Position der Karte gewechselt und haben diese dann nicht mehr bewegt. Wie funktioniert denn nun das? Das Geheimnis liegt bei der Erdanziehung. Du spürst diese beim Treppensteigen oder wenn du einen Stein wirfst. Alles wird mit von einer unsichtbaren Kraft vom Erdmittelpunkt angezogen. Diese Kraft nennt man Gravitationskraft. Ohne sie würden wir herumschweben oder in den Weltraum entschwinden. Die Gravitation zieht Körper an und damit letztlich natürlich auch unseren Sensor, der diese Kraft immerwährend wahrnimmt. Damit können wir herausfinden, wo oben und unten ist und damit die Position der Karte bestimmen, auch wenn wir eigentlich nur Beschleunigungen messen können.

Der Beschleunigungssensor kann drei verschiedene Achsen messen. Diese haben wir dir in dieser Grafik zusammengestellt:

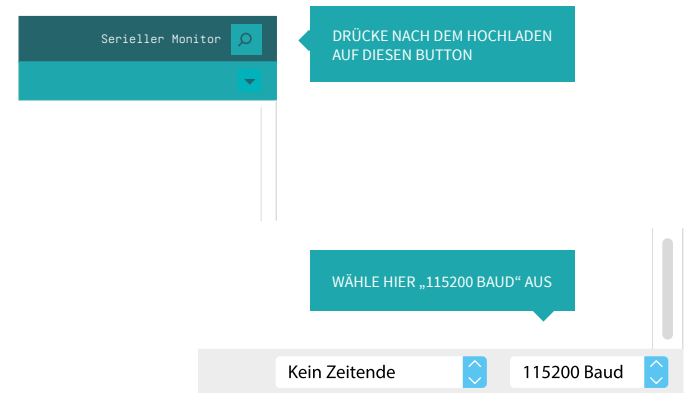


Es gibt eine X-, Y- und eine Z-Achse. Wenn du die OXOcard vor dir hast und sie nach links und rechts kippst, rotierst du die Karte um die X-Achse, wenn du sie nach vorne und zurück kippst, ist das die Y-Achse und wenn du sie um dies nach links und rechts drehst, ist das die Z-Achse.

Schauen wir uns nun ein weiteres Beispiel an:

 Lade das Beispiel `sample10c` und übermittle es auch auf die Karte.

Dann öffnest du den seriellen Monitor, wie auf unterem Bild dargestellt, und wählst in der unteren rechten Ecke „115200 Baud“ aus. Nimm nun die Karte in die Hand und neige sie ein wenig von links nach rechts. Schau dir nur die Zahlen auf dem Computer an. Was stellst du fest?



Das Fenster auf dem Computer nennt man einen seriellen Monitor. Du kannst damit Daten von deiner OXOcard auf den Computer schicken und diese dort anzeigen lassen. In unserem Fall haben wir den Beschleunigungswert um die X-Achse ausgegeben. Die Anzeige zeigt jeweils pro Zeile einen Wert zwischen -2000 und +2000 an. Beispielsweise sieht das so aus:

```
x = 0
x = 12
x = 36
x = 12
x = 24
x = 12
```

Wenn du die Karte waagrecht in der Hand hältst, zeigt der Sensor einen Wert zwischen -50 und 50 an. Wenn du sie dann leicht nach links und rechts kippst, erhöht sich die Zahl auf ca. 255, wenn du sie im Uhrzeigersinn drehst, bzw. ca. -255, wenn du sie im Gegenuhrzeigersinn drehst. Die Zahl ändert laufend ein wenig, da der Sensor sehr empfindlich ist und bereits kleinste Veränderungen wahrnimmt.

Schauen wir uns das Programm an:

```
#include „OXOCardRunner.h“
```

```
void setup() {
}
```

```
void loop() {
  int x = getXAcceleration();
  b = 10;
```

DIESE FUNKTION LIEFERT DIE BESCHLEUNIGUNG UM DIE X-ACHSE.

```
print(“x = ”);
println(x);

delay(200);
}
```

«PRINT» SCHREIBT EINEN WERT IN DAS FENSTER AUF DEINEM COMPUTER.

«PRINTIN» DRUCKT EINEN WERT UND WECHSELT AUF EINE NEUE ZEILE.

Die erste Zeile im `loop` beginnt mit dem Befehl `getXAcceleration()`;

```
int x = getXAcceleration();
```

Solche Befehle kennst du ja schon. Dieser hier liefert einen Wert zwischen ca. -2000 und +2000. Wie oben bereits geschrieben, ist eine positive Zahl eine Beschleunigung im Uhrzeigersinn, bzw. negativ im Gegenuhrzeigersinn. 255 bzw. -255 entspricht ungefähr der Erdbeschleunigung, wenn du die Karte mit der Kante nach unten hältst. Wenn du die Karte rasch bewegst, d.h. mehr Beschleunigung aufwendest, wird der Wert höher.

Wir möchten nun wieder etwas zeichnen und schauen uns hierzu das nächste Beispiel an.



Lade und starte das Beispiel `sample10d`.

Bewege die Karte und versuche zu verstehen, was das Programm macht.

Es wird wieder die Beschleunigung der Karte gemessen. Wenn du diese waagrecht hältst und nach links und rechts kippst, wird auf der entsprechenden Seite eine Linie gezeichnet.

Wir haben zwei Bedingungen programmiert:

```
if (x > 50) {  
    drawLine(0,0,0,7);  
}
```

Wenn der Wert grösser als 50 ist, zeichnen wir eine Linie am rechten Rand. Umgekehrt zeichnen wir eine Linie am linken Rand, wenn $x < -50$ ist. Wenn die Karte waagrecht liegt, wird nichts gezeichnet. Voila! Jetzt kannst du dir vermutlich viel besser vorstellen, wie die Programmierer auf dem Smartphone mit dem Beschleunigungssensor ein Spiel steuern können. 😊

AUFGABE

Jetzt wird's knifflig. Wir möchten dich jetzt beauftragen, das Programm `sample10d` so anzupassen, dass es die Balken unten und oben an der Karte anzeigt, wenn du die Karte nach vorne oder nach hinten neigst. Dafür brauchst du einen neuen Befehl, der `getYAcceleration()` heisst und ansonsten genau gleich funktioniert wie `getXAcceleration()`, jedoch die Beschleunigung um die Y-Achse misst.

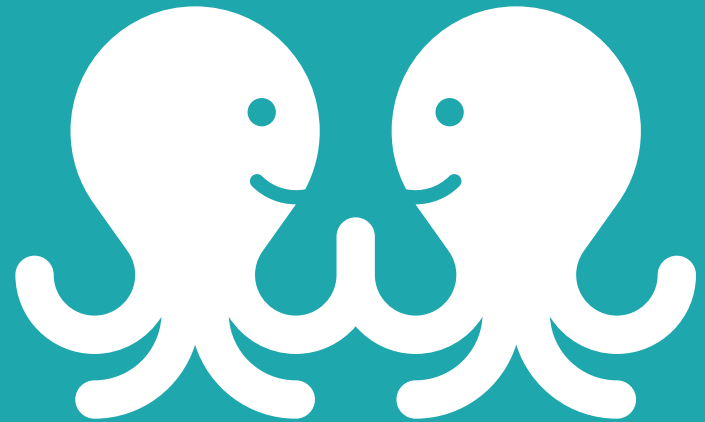


Öffne das Programm „sample10d“

LÖSUNG - SEITE 183



ZUSAMMEN
MACHT'S
MEHR SPASS



KOMMUNIKATION IST ALLES

Ohne den Austausch mit anderen ist das Leben trist. Deshalb gibt es verschiedene Möglichkeiten, wie du mit der OXOcard kommunizieren kannst. In den vorigen Kapiteln haben wir Buttons benutzt, um etwas auf der Karte auszulösen. Zudem kann die Karte uns über die LED's und den Tongenerator Informationen mitteilen. Eine weitere Möglichkeit des Datenaustauschs findet über Kabel statt. Wenn du das ganze Büchlein bis hierhin durchgearbeitet hast, kennst du den seriellen Monitor, über den du mittels `print` Daten an den Computer schicken kannst. Das funktioniert übrigens auch umgekehrt, wie du in der Arduino-Dokumentation auf arduino.cc nachlesen kannst.

Die nächste Stufe der Kommunikation erfolgt über Funk und ist für uns Menschen unsichtbar. Vieles wird heute über Funk übertragen. Beim Smartphone haben wir gleich mehrere Funksender und -empfänger, mit denen wir Daten austauschen können. zum Beispiel die Datenverbindung des Telekomanbieters, das WiFi-Netz zu Hause und wenn du kabellose Kopfhörer hast, gibt es da noch Bluetooth. Für letztere Technik haben wir uns bei der OXOcard entschieden und ein Bluetooth-Funkmodem integriert.

Modem ist die Abkürzung für Modulation und Demodulation und bedeutet auf deutsch einfach, dass dieses Gerät Daten in eine Funkwelle überträgt („moduliert“) und sie am andere Ende wieder in Daten zurückwandelt, eben „demoduliert“. Unsere OXOcard ist in der Lage, über das Bluetooth-Funknetz Daten zu empfangen und zu verschicken und das beste daran ist, dass dies für dich ganz einfach ist.

Damit du dieses Kapitel durchspielen kannst brauchst du zwei OXOcards. Wenn ihr die Aufgabe in der Schule macht, ist jetzt der späteste Zeitpunkt, sich einen Kameraden oder eine Kameradin zu suchen, der/die mit dir die Welt der Kommunikation entdecken möchte.

Bluetooth

Deine OXOcard verfügt über ein Bluetooth-Modem. Bluetooth oder zu deutsch Blauzahn war ein dänischer Wikingerkönig, der für seine Kommunikationsfähigkeit bekannt war und laut Wikipedia der Namensvetter der Technologie ist. Bluetooth wurde hauptsächlich durch die Firmen Ericsson und Nokia entwickelt, ist aber ein sogenannter Industriestandard, bei dem viele Firmen weltweit mitmachen, indem sie Funkchips und Geräte entwickeln. Nebst Funkkopfhörern finden wir die Bluetooth-Funktion auch in Computermäusen oder Funktastaturen, d.h. alles Geräte, die über kurze Distanz Daten austauschen. Bluetooth eignet sich deshalb so gut für diese Anwendungen, weil es extrem wenig Strom braucht und auch mit kleinen Batterien sehr lange Laufzeiten erlaubt.

Wir haben für dich ein einfaches Beispiel vorbereitet, das wir nun gemeinsam anschauen wollen.

„ICH BIN DIE OXOCARD 1 UND ICH BIN HIER!“

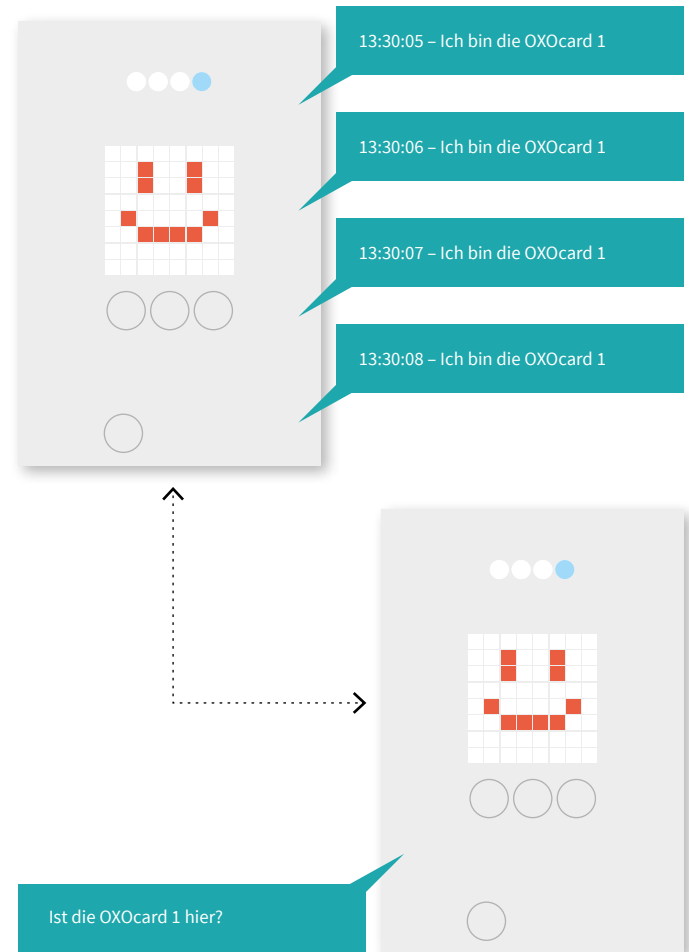
Du brauchst für dieses Programm zwei Karten. Die eine Karte ist der Sender, die andere Karte der Empfänger der Nachricht.

↑ Lade `sample11a` auf die Sender-Karte, das Beispiel `sample11b` auf die Empfänger-Karte. Starte die beiden Programme.

Beide Karten werden nun miteinander eine Information austauschen. Nach 30 Sekunden stellt sich die Senderkarte automatisch ab. Du kannst sie wieder starten, indem du den Reset-Button (gleich neben dem USB-Stecker) drückst. Was stellst Du fest?

Solange die Senderkarte in der Nähe ist und läuft, wird das Herz auf der Empfängerkarte stärker beleuchtet.

Zur näheren Erklärung zuerst ein Bild.



Die Senderkarte auf der linken Seite ist als „Plapper-
maul“ programmiert. Es schickt in kurzen Abständen die
Information, dass sie „online“ ist und die Karte Nummer 1
ist. Diese Art der Kommunikation, die Apple vor ein paar
Jahren erfunden hat, heisst `iBeacon`. „beacon“ heisst auf
Deutsch „Leuchtturm“. Wie bei einem Leuchtturm schickt
die Karte in regelmässigen Abständen ein Signal. Die
Empfänger können das Signal nun empfangen und lesen,
welche Karte das verschickt hat.

Schauen wir uns als nächsten den Code an (sample11a):

```
#include „OX0CardRunner.h“

void setup() {
  setupAsIBeacon(1);
  clearDisplay();
  drawNumber(1);
}

void loop() {
  handleAutoTurnOff(30);
}
```

```
setupAsIBeacon(1);
```

Diese Anweisung macht aus unserer Karte einen virtuellen
Leuchtturm und schickt hierbei die Nummer 1 als
Funksignal.

Die Anweisungen

```
clearDisplay();
drawNumber(1);
```

schreiben die Zahl 1 auf das Display, damit wir wissen,
dass dies unser Sender ist und

```
handleAutoTurnOff(30);
```

im Loop-Block (wichtig!), schaut bei jedem Durchgang, ob
schon 30 Sekunden um sind und sobald das erreicht ist,
wird die Karte abgestellt. Du kannst sie jederzeit erneut
starten, indem du den Reset-Button betätigst.

Schauen wir uns den Empfänger-Code an (sample11b):

```
void setup() {
}

void loop() {
  if (findBeacon()) (

      drawImage( 0b00000000,
                 0b01101100,
                 0b00000000,
                 0b00000000,
                 0b01000100,
                 0b00111000,
                 0b00000000,
                 0b00000000);
```

```

    delay(1000);
  } else {
    drawImage( 0b00000000,
              0b11101110,
              0b10101010,
              0b00000000,
              0b01000100,
              0b00111000,
              0b00000000,
              0b00000000)
            20);
  }
}

```

```

(findIBeacon(1)) {
  ...
} else {
  ...
}

```

Die Funktion, die wir hier benutzen heisst `findIBeacon` und du musst lediglich angeben, welche Karte du suchst – in dem Fall die Karte Nummer 1. Sobald die Karte in Reichweite ist, wird der erste Block in geschweiften Klammern ausgeführt. Andersfalls kommt der Else-Block zum Zug. Wir können jetzt also in unserem Programm eine Weiche stellen, wenn eine bestimmte Karte in der Nähe ist! 😊

Du stellst vielleicht fest, dass die Kommunikation manchmal ein paar Sekunden hinterherhinkt, bzw. die Karte nicht erkannt wird. Das hat damit zu tun, dass es in unserem Umfeld heute sehr viele Funksignale gibt, was dazu führen

kann, dass es Überschneidungen gibt und dabei auch Fehler bei der Übertragung stattfinden. Du kannst dir das ungefähr so vorstellen wie an einer lauten Veranstaltung, wo sich Leute unterhalten wollen, es aber aufgrund des Lärmpegels nicht immer klar ist, was dein Gegenüber sagen wollte und man es manchmal zwei oder dreimal wiederholen muss. In der Computertechnik werden solche Übertragungsfehler in der Regel durch geschickte Programmierungen vom Benutzer verborgen.


Wir kommen nun zum letzten Beispiel und einem neuen Sprachbaustein, den jeder Programmierer kennt und wir dir natürlich nicht vorenthalten wollen, die Subroutine. Man nennt sie auch Funktion oder Prozedur. Letztlich ist es alles dasselbe und erlaubt dir, häufig genutzte Teile unter einem Namen zu speichern. Hierzu gibt's sofort ein einleuchtendes Beispiel.

DAS WIEDERVERWENDBARE HERZ

Wenn du in deinen Beispielen viele Herzen zeichnen möchtest, musst du jedes Mal einen Block in der Form programmieren:

```
drawImage(0b01100110,  
          0b11111111,  
          0b11111111,  
          0b11111111,  
          0b01111110,  
          0b00111100,  
          0b00011000,  
          0b00001000);
```

Beispielsweise haben wir im Empfängerbeispiel zwei Herzen gezeichnet. Eines, das stark leuchtet, wenn die andere Karte in der Nähe ist, und eines, das nur schwach leuchtet, wenn wir sie nicht empfangen können. Wir möchten dir nun zeigen, wie man solche sich wiederholende Blöcke vermeiden kann.

 Starte auf der Empfängerkarte das Programm `sample11c` und schau dir den Code an.

Du siehst offenkundig vom Ablauf her keine Veränderung gegenüber der vorigen Version, obwohl wir das Herz nur noch an einer Stelle programmiert haben. Das Zeichnen des Herzens wurde jedoch in eine Subroutine ausgelagert, die `drawHeart` heisst.

Um zu verstehen, wie Subroutinen funktionieren, schauen wir uns das Beispiel in zwei Teilen an. Im ersten Teil siehst du, wie wir `drawHeart` nutzen:

```
#include „OX0CardRunner.h“  
  
void setup() {  
}  
  
void loop() {  
  if findIBeacon(1){  
    drawHeart(225);  
    delay(1000);  
  } else {  
    drawHeart(20);  
  }  
}
```

DRAWHEART IST EINE SUBROUTINE, DIE EIN HERZ ZEICHNET.

HIER BRAUCHEN WIR SIE NOCHMALS.

Wenn wir das `iBeacon` sehen, soll das Herz mit maximaler Leuchtstärke erstrahlen (255), wenn wir es nicht mehr sehen, reduzieren wir die Leuchtkraft auf 20. Die Anwendung entspricht normalen Funktionsaufrufen, wie du sie an verschiedene Stellen schon gebraucht hast. Effektiv hast du bereits viel Erfahrung mit der Nutzung solcher Subroutinen. Beispielsweise ist `delay` eine oder `find1Beacon`.

Im folgenden Programmteil erstellen wir nun eine eigene Subroutine.

HIER STEHT EIN
BELIEBIGER NAME

HIER KÖNNEN WIR BELIEBIGE PARAMETER ODER DIE KLAMMERN
LEER LASSEN, WENN WIR KEINE WERTE BRAUCHEN.

```
void drawHeart(byte brightness) {  
    if (findBeacon()) (  
  
        drawImage( 0b00000000,  
                   0b01101100,  
                   0b00000000,  
                   0b00000000,  
                   0b01000100,  
                   0b00111000,  
                   0b00000000,  
                   brightness);  
    }  
}
```

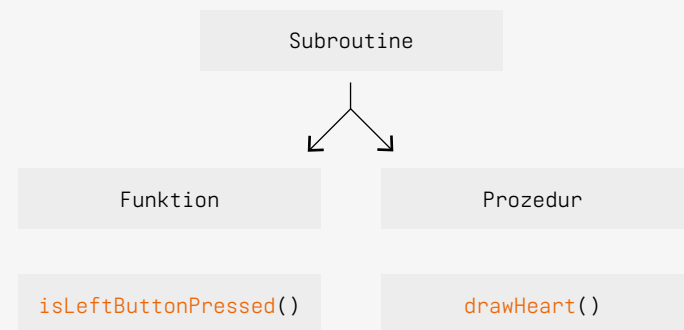
Die Anweisung

```
void drawHeart(byte brightness) {  
}
```

definiert eine neue Subroutine, die einen Parameter `brightness` braucht. Innerhalb der geschweiften Klammern `{}` kannst du nun schreiben, was du möchtest. Damit definierst du eine neue Subroutine, die du fortan überall in deinem Programm brauchen kannst.

SUBROUTINE, FUNKTION ODER PROZEDUR?

Es gibt verschiedene Begriffe, die alle dasselbe meinen, wengleich mit unterschiedlicher Bedeutung. Wenn du dich fürs Programmieren interessierst, wirst du alle diese Namen hören, daher hier ein paar Worte dazu. Subroutinen kapseln einen Codeabschnitt in einem wiederverwendbaren Block. Die Routinen können entweder parameterlos sein oder eine Anzahl von Parametern erfordern. Wenn die Subroutine einfache Anweisungen ausführt, spricht man von einer einfachen Prozedur. Wenn sie nach Fertigstellung einen Wert zurückliefert, spricht man von einer Funktion. Beispielsweise ist `delay` eine einfache Prozedur, wohingegen `findIBeacon` eine Zahl zurückgibt. Wenn diese „0“ ist, ist das `iBeacon` nicht in der Nähe bzw. ausgeschaltet.



SPIEL „METEORIT“



METEORIT

Mit diesem Kurs können wir dir nicht alle Aspekte der OXOcard oder der Programmiersprache C++ zeigen, daher haben wir uns auf die interessanten und wichtigsten Teile beschränkt, die dir ein Grundverständnis geben, aber auch schon so viel Können aufbauen, dass du ein kleines Spiel programmieren kannst.

Für diese Zwecke haben wir dir ein kleines Spiel vorbereitet, das du nach deinem Geschmack abändern/erweitern kannst.

Das Spiel heisst „Meteor“ und du steuerst ein kleines Raumschiff, das sich in einen Meteoritenschwarm verirrt hat. Deine Aufgabe ist es, das Raumschiff ohne Kollisionen zwischen den Meteoriten zu navigieren. Das Spiel endet, wenn du einen Meteoriten triffst. Steuere mit dem linken und rechten Button, mit dem mittleren Button startest du das Spiel neu.

AUFGABE bzw. Einladung zum Spielen:



Starte das Beispiel `sample12` und spiele damit ausgiebig. 😊



AUF DER FOLGENDEN SEITE ERKLÄREN
WIR DIR DEN PROGRAMMAUFBAU
UND ZEIGEN DIR, WO DU EIGENE
ERWEITERUNGEN VORNEHMEN KANNST.

Die folgende Darstellung zeigt den gesamten Programmcode des Meteoritenspiels. Er hilft dir einerseits, das Wesentliche des Spielaufbaus besser zu erfassen, andererseits gibt er dir die Möglichkeit, mit deinen Erweiterungen eine coole eigene Variation zu programmieren.

A

```
#include „OXOCardRunner.h“

void setup() {
    clearDisplay();
}

byte ship_x = 0;
byte ship_y = 7;
byte meteor_x = 0;
byte meteor_y = 0;
bool stopped = false;

void loop() {
```

Wir haben bei dem Spiel einen Meteor und ein Raumschiff. Die Variablen `ship_x` und `ship_y` speichern die jeweilige Position des Raumschiffs, in `meteor_x` und `meteor_y` wird die Position des Meteors gespeichert. Beim Spielen hast du bemerkt, dass das Spiel stoppt, wenn du getroffen bist. Wenn das der Fall ist, wird die Variable `stopped` auf `true` gesetzt.

B

```
handleAutoTurnOff(120);
```

Diese Anweisung stellt den Computer nach 120 Sekunden ab. Du kannst die Zahl ändern, um das Gerät früher oder später in den Schlafmodus zu versetzen.

C

```
if (stopped) {
    if (isMiddleButtonPressed()) {
        stopped = false;
    } else {
        resetTimer();
        drawIntro();
        return;
    }
}
```

Wir sind hier in der Loop-Schleife. Als erstes testen wir, ob wir „gestoppt“ sind. In dem Fall fragen wir die mittlere Taste ab. Falls du draufdrückst, sind wir nicht mehr gestoppt, d.h. `stopped` wird auf `false` gesetzt und damit wird der Programmablauf fortgesetzt. Im `Else`-Fall hast du einen neuen Befehl, der `return` heißt. Er stoppt die Ausführung von `Loop` und beginnt dann wieder von vorne.

```
D clearDisplay();
drawPixel(meteor_x, meteor_y, 80);
drawPixel(ship_x, ship_y);
drawPixel(ship_x + 1, ship_y);
```

Wenn wir also nicht gestoppt sind, geht's weiter. Als nächstes zeichnen wir den Meteor und dann das Raumschiff. Letzteres besteht aus zwei nebeneinanderliegenden Pixeln, daher zeichnen wir noch einen Pixel an der Koordinate `ship_x + 1, ship_y`.

```
E if ((ship_x == meteor_x || ship_x + 1 == meteor_x)
    && ship_y == meteor_y) {

    noTone();
    drawGameOver();

    ship_x = 0;
    ship_y = 7;
    stopped = true;
}
```

Diese `if`-Weiche fragt ab, ob die Koordinate des Raumschiffs derjenigen des Meteors entspricht. In dem Fall haben wir eine Kollision und das Spiel ist zu Ende. Wir zeichnen ein Rechteck, lassen einen tiefen Ton spielen und dann setzen wir die Spielvariablen wieder in den Ursprung. Hier setzen wir auch `stopped` wieder auf `true`, damit wir oben bei C wieder abfragen können, ob das Spiel gestoppt ist und dann entsprechend handeln können.

```
F if (isLeftButtonPressed()) {
    if (ship_x > 0) {
        ship_x = ship_x - 1;
    } else {
        ship_x = 0 ;
    }
}
```

Wenn der linke Button gedrückt ist, schieben wir das Schiff nach links, ausser wir sind schon am Rand.

G

```
if (isRightButtonPressed()) {
  if (ship_x < 6) {
    ship_x = ship_x + 1;
  } else {
    ship_x = 6;
  }
}
```

Wenn wir den rechten Button drücken, schieben wir das Schiff nach rechts bis zum Rand.

H

```
meteor_y = meteor_y + 1;

if (meteor_y > 7) {
  meteor_x = random(8);
  meteor_y = 0;
}
```

Hier bewegen wir den Meteor um eine Position nach unten. Wenn der Meteor unten angekommen ist, legen wir eine neue Ausgangsposition mit der Zufallsfunktion `random` fest und beginnen wieder von oben.

I

```
delay(30);
}
```

Dieser Delay-Aufruf stellt sicher, dass das Spiel nicht zu schnell wird.

Erweiterungen

Wie du beim Spielen sicherlich festgestellt hast, haben wir es relativ einfach gehalten. So gibt es nur einen Crash-Ton, sonst haben wir auf Akustik verzichtet. Die Darstellungen sind sehr einfach gehalten und es gibt auch keine Levels. All dies kannst Du jetzt selber einbauen.

Die folgenden Anweisungen sind Beispiele und zeigen, wie du bestimmte Ideen in die Realität umsetzen kannst. Sie sind weder vollständig, noch besonders ausgefeilt. Wir möchten dich damit animieren, selber zu experimentieren, damit du deine eigenen Erfahrungen machen kannst.

Gib mir einen Beep!

Mit dieser Erweiterung hörst du einen regelmässigen Beep, quasi eine Art Herzschlag des Spiels. Diese Erweiterungen geht so:

Ergänze im Block A einen Variable:

```
byte loop_counter = 0;
```

Diese Variable zählt die Durchgänge des Loop-Blocks und wir brauchen Sie bei dieser Passage, die du zwischen H und I einsetzt:

```
if (loop_counter < 10) {
  loop_counter = loop_counter + 1;
} else {
  loop_counter = 0;
  tone(1000, 50);
}
```

Dieser Block wird bei jedem Durchgang, das heisst alle 50 Millisekunden einmal ausgerufen. Wenn die Variable `loop_counter` noch nicht 10 ist, wird sie um eins erhöht. Wenn wir 10 erreicht haben, wird die Variable wieder auf 0 gesetzt, damit sie wieder bis 10 zählen kann und ein Signalton wird ausgeführt.

Damit der Crash-Ton in jedem Fall sauber gespielt wird, müssen wir folgende Zeile

```
noTone();
```

im Block E gleich vor der Zeile `tone(100,1000);` ergänzen.

Du kannst hier folgendes ändern: wenn du die Zahl 10 verringerst, wird der Beep schneller, umgekehrt langsamer. Bei der `tone`-Funktion kannst du einen beliebigen Ton hinterlegen, der dir gefällt.

Schneller!

Viele Spiele werden schneller, je länger man sie spielt. Das möchten wir natürlich auch machen. Hierzu sind folgende Ergänzungen zu machen:

Im Block C schreibst du vor dem `return`-Befehl folgendes:

```
resetTimer();
```

Ersetze nun den I-Block, d.h. die Delay-Anweisung mit folgendem Code:

```
int seconds = getTimerSeconds();
if (seconds < 5) {
    delay(50);
} else if (seconds < 10) {
    delay(40);
} else {
    delay(30);
}
```

Wir verwenden hier zwei neue Befehle `resetTimer` und `getTimerSeconds()`. Der Befehl `getTimerSeconds()` gibt die Anzahl der Sekunden zurück, seit das Gerät gestartet wurde, bzw. seit dem letzten Aufruf von `resetTimer`. Die Anweisungen machen nun folgendes. Bei jedem Neustart bzw. Fehler setzen wir den Timer wieder zurück. Im zweiten Block fragen wir ab, wie viele Sekunden bereits vergangen sind. Unter 5 Sekunden warten wir 50 Millisekunden, zwischen 5 und 10 Sekunden warten wir etwas weniger lang und wenn es über 10 Sekunden dauert, geben wir uns noch noch ganz kurze Pausen, mit dem Effekt, dass das Spiel immer schneller wird.

Du kannst hier die Intervalle ändern und natürlich auch die Geschwindigkeit. Du kannst nach demselben Muster auch weitere Abstufungen einbauen, z.B. 50, 40, 30, 20.

Spiel-Intro

Jedes Spiel hat ein Intro, d.h. eine Einleitung, die abgespielt wird, bevor das Spiel startet. Das wollen wir jetzt auch machen und ändern folgendes ab.

Ersetze im A-Block die Zeile

```
bool stopped = false;
```

mit

```
bool stopped = true;
```

Ergänze im C-Block vor dem `return` folgende Zeile:

```
drawIntro();
```

Das ist ein Funktionsname für eine Funktion, die wir ganz am Schluss des Codes deklarieren wollen. Hier findest du den Code dazu:

```
void drawIntro() {
    clearDisplay();
    drawImage(0b11100111,
              0b10000101,
              0b10100101,
              0b11100111,
              0b00000000,
              0b00000000,
              0b00000000,
              0b00000000);
    delay(300);
    clearDisplay();
    drawImage(0b00000000,
              0b00000000,
              0b00000000,
              0b00000000,
              0b00000000);
}
```



```

        0b01111110,
        0b00111100,
        0b00011000,
        80);
    delay(300);
}

```

Dieser Block zeichnet zwei Bilder, die aus Nullen und Einsen aufgebaut sind. Du kannst sie erkennen, wenn du dieses Blatt etwas weiter vom Kopf weghältst und die Augen etwas zusammenkneifst. `clearDisplay` löscht den Bildschirm, `drawImage` zeichnet das Bild, `delay` wartet die Anzahl Millisekunden, die du angibst.

Du kannst folgendes anpassen: Du kannst weitere Bilder ergänzen, bzw. die bestehenden abändern. Du kannst natürlich auch Geräusche mit dem `tone`-Befehl ergänzen.

Eigene Crash-Animation

Nach demselben Prinzip wie die Intro-Animation aufgebaut ist, möchten wir nun eine Crash-Animation programmieren, die dann abgespielt wird, wenn das Raumschiff mit dem Meteor zusammenstösst.

Ersetze bitte folgende Zeilen im Block E:

```

drawRectangle(0, 0, 8, 8, 255);
noTone();
tone(100, 1000);

```

Mit folgender Zeile:

```

noTone();
drawGameOver();

```

(Wenn Du das Beep-Beispiel nicht gemacht hast, fehlt dir möglicherweise die Zeile `noTone();`.)

Dann fügst du ganz unten am Programm folgende Subroutine ein:

```

void drawGameOver() {
    for (int i = 0; i < 1000; i++) {
        byte x = random(8);
        byte y = random(4);
        byte b = random(255);
        drawPixel(x, 4+y, b);
        tone(random(200));
    }
    noTone();
}

```

Die Funktion zeichnet unterschiedliche Pixel in unterschiedlicher Helligkeit und dazu wilde Töne! Lass deiner Kreativität freien Lauf.



Du magst die Änderungen nicht abtippen?
Kein Problem. Du findest alles in Beispiel [sample12b](#).

WAS KOMMT
ALS NÄCHSTES?



Dieser Kurs kann dir natürlich nicht alle Aspekte der Programmierung zeigen und ist als Einstieg in die Welt der Informatik gedacht. Wenn du mehr wissen möchtest, empfehlen wir dir folgende Webseiten:

www.oxocard.ch

Dies ist die offizielle Seite der OXOcard. Du kannst allgemeine Informationen zur Karte anschauen, sämtliche Beispiele runterladen und eine Menge an Hilfestellungen und Zusatzinformationen finden.

<https://www.arduino.cc/>

Die OXOcard ist Arduino-kompatibel. Arduino ist ein Open-Source-Projekt, an dem viele Leute auf der ganzen Welt arbeiten, um dir und vielen Millionen anderer Programmier- und Technikbegeisterten den Einstieg in diese faszinierende Welt einfacher zu gestalten. Du findest hier viele zusätzliche Beispiele, die wir aufgrund der begrenzten Platzverhältnisse in diesem Büchlein weggelassen haben.

www.google.com

Google muss man eigentlich nicht noch gesondert auführen, da du Google sicherlich schon rege für andere Aufgaben nutzt. Hier nur ein Tipp: wenn du Beispiele suchst, die du auf der OXOcard nutzen möchtest, gib zusätzlich zu deinen Suchbegriffen „Arduino“ ein. Das führt dazu, dass Google dir Beispiele nennt, die bereits für Arduino konzipiert sind. Man kann sicherlich nicht alle eins zu eins verwenden, jedoch gibt es sehr viele verschiedene Programmiersprachen, deren Code du nicht ohne grössere Anpassungen und Fachwissen auf unsere Karte bringst.

GLOSSAR

In dieser Liste findest du Erklärungen zu technischen Begriffen, die wir in dem Booklet verwendet haben.

ASSEMBLER - PROGRAMMIERSPRACHE

Dies ist eine sehr einfache, maschinennahe Programmiersprache, bei der der Programmierer anstelle von Bits sehr einfache Befehle zur Verfügung hat. Assembler wird nur noch von Spezialisten geschrieben und ist sehr anspruchsvoll. Assembler wird heute hauptsächlich von Programmierern von Betriebssystemen und Compilerbauern gebraucht.

BIT

Ein Bit ist die kleinste speicherbare Einheit des Computers. Ein Bit kann entweder 0 oder 1 sein.

BYTE

Ein Byte ist eine Reihe von 8 bits. Wenn du alle möglichen 0-1-Kombinationen ausschreibst, kommst du auf 256 verschiedene Varianten. Daher kann ein Byte 256 unterschiedliche Werte speichern (0-255).

COMPILER

Ein Compiler ist ein Programm, welches ein menschenlesbares Computerprogramm in eine maschinenlesbare Form umwandelt.

Das Beispiel `sample1`, welches in der Programmiersprache C++ geschrieben ist und 148 Zeichen umfasst, wandelt der Compiler in ein Maschinenprogramm mit 36'960 Bytes um!

FUNKTION

Eine Funktion sammelt bestimmte Befehle zu einem wiederverwendbaren Konstrukt. Eine Funktion hat immer einen Namen, kann Funktionsparameter erwarten und auch einen Rückgabewert liefern.

iBEACON

Das ist eine Erfindung von Apple, bei der es darum geht, dass ein Gerät regelmässig eine eindeutige Nummer funkt. Wenn man weiss, an welchem Standort das Gerät mit der Nummer steht, kann man durch den Empfang der Nummer erkennen, wo man sich aktuell befindet. iBeacons werden in der Regel zur Positionsangabe innerhalb von Gebäuden genutzt.

LED

Light Emitting Diode – dt. Leuchtdiode. Fliesst durch sie Strom, strahlt sie Licht aus.

MASCHINENSPRACHE

Ein Computer enthält eine Sammlung sehr einfacher Befehle, die aus Bit-Mustern bestehen und die man Maschinensprache nennt. Es gibt verschiedene Maschinensprachen, jedoch sind die meisten aus Bits aufgebaut. Einfache Computer haben 8bit-Befehle, komplexere Computer, wie diejenigen in deinem Handy oder Computer können 32- oder 64-bit-Befehle enthalten und sind entsprechend umfangreicher und leistungsfähiger.

PARAMETER

Griechisch: para, „neben“ und metron, „Mass“. In der Informatik wird damit eine spezielle Variable bezeichnet, welche einer Funktion einen Wert gibt.

PIXEL

Ein Pixel ist ein Lichtpunkt auf einem Bildschirm. Je nach Definition spricht man bei hochauflösenden Bildschirmen von einem einzelnen Farbpixel, der aber eigentlich aus drei kleineren Pixeln für die Farben rot, grün und blau besteht.

SENSOR

Lateinisch: sentire, „fühlen“, „empfinden“. Ein Sensor ist ein elektronisches oder mechanisches Gerät, welches etwas Bestimmtes wahrnimmt, z.B. Licht, Druck, einen chemischen Stoff, etc.

SUBROUTINE

Ist eine Funktion oder eine Prozedur, welche eine Menge an Befehle zu einem wiederverwendbaren Block zusammenfasst und diesem einen Namen gibt. Siehe auch „Funktion“ und „Prozedur“.

VARIABLE

lat. lateinisch variare „verändern“ Dies ist der sprechende Namen für einen Speicherplatz auf deinem Computer. Man kann unterschiedliche Werte in Variablen speichern, d.h. Zahlen, Bilder, Texte und auch Listen.

INT ODER INTEGER

Ist ein Zahlentyp, der in unserem Fall eine Zahl zwischen -32768 bis +32768 abbilden kann. 32768 ist 2^{15} oder die Anzahl aller möglichen Kombination von Nullen und Einsen, wenn du 15 davon in eine Reihe bringst. Ein Integer-Wert mit Vorzeichen braucht 16 Bits. Das erste Bit speichert, ob es eine positive oder negative Zahl ist.

PROGRAMMIERSPRACHE

Eine Programmiersprache dient zur Programmierung von Computern.

PROZEDUR

Ist eine Subroutine, welche eine Menge an Befehlen ausführt.

ALLE OXOCARD-BEFEHLE AUF EINEN BLICK



DIE OXOCARD IST EIN ARDUINO-KOMPATIBLER COMPUTER.

Für Arduino gibt es auf dem Internet eine riesige Menge an Beispielprogrammen, die du runterladen und meist mit wenigen Anpassungen für die Oxocard nutzbar machen kannst.

Die wichtigste Quelle ist natürlich unsere Seite www.oxocard.ch.

Daneben findest du eine Menge an Beispielen auf der Homepage des Arduino-Projektes, auf dem unsere Karte basiert: www.arduino.cc.

Unter <https://www.arduino.cc/en/Reference/HomePage> findest du sämtliche Befehle der Arduino-Umgebung, die du alle auch für die Karte nutzen kannst.

Damit es für Programmieranfänger einfacher wird, haben wir folgende Befehle zusätzlich für dich vorbereitet. Damit diese verfügbar sind, musst du auf der ersten Zeile des Arduino-Programms folgendes einfügen:

```
#include „OXOCARDRunner.h“
```

Danach kannst du folgende Befehle nutzen: (Hinweis: kursiv gedruckte Parameter können weggelassen werden.)

clearDisplay();
Löscht alle Pixel.

drawChar(x, y, c, brightness)
Zeichnet ein Zeichen an der Stelle x,y:

```
drawChar(4,4,'a');
```

bzw.

```
drawChar(4,4,'a',100);
```

drawCircle(x, y, r, brightness)
Zeichnet einen Kreis mit dem Mittelpunkt bei x,y und dem Radius r. Beispiel:

```
drawCircle(4,4,2);
```

oder

```
drawCircle(4,4,2,100);
```

drawDigit(x, y, digit, brightness)
Zeichnet eine Ziffer zwischen 0 und 9 an der Stelle x,y:

```
drawDigit(4,4,3);
```

bzw.

```
drawDigit(4,4,3,100);
```

drawFilledCircle(x, y, r, brightness)

Identisch wie drawCircle, jedoch wird das Rechteck ausgefüllt.

drawFilledRectangle(x, y, w, h, brightness)

Identisch wie drawRectangle, jedoch wird das Rechteck ausgefüllt.

drawFilledTriangle(x0, y0, x1, y1, x2, y2, brightness)

Identisch wie drawTriangle, jedoch wird das Dreieck ausgefüllt.

drawImage(b0,b1,b2,b3,b4,b5,b6,b7,brightness)

Zeichnet in Pixelbild, indem du die Pixel direkt drawImage übergibst:

```
drawImage(0b00000000,  
          0b00101000,  
          0b00101000,  
          0b00000000,  
          0b01000100,  
          0b00111000,  
          0b00000000,  
          0b00000000);
```

drawImage(image[8], brightness)

Zeichnet ein Pixelbild, wobei du dieses in Form einer Array-Variable übergeben kannst:

```
byte smiley[8] = {  
    0b00000000,  
    0b00101000,  
    0b00101000,  
    0b00000000,  
    0b01000100,  
    0b00111000,  
    0b00000000,  
    0b00000000  
};  
  
drawImage(smiley);
```

drawLine(x0, y0, x1, y1, brightness)

Zeichnet eine Linie von x0,y0 zu x1,y1. Beispiel:

```
line(0,0,7,7);
```

oder

```
line(0,0,7,7,100);
```

drawNumber(number, brightness)

Zeichnet eine Zahl zwischen 0 und 99 auf den Bildschirm.

```
byte irgendwas = 10;  
drawNumber(irgendwas);
```

drawPixel(x, y, brightness)

Lässt einen Pixel an der Koordinaten x,y leuchten.
Beispiele:

```
drawPixel(2,2)
```

oder

```
drawPixel(2,2,100);
```

drawRectangle(x, y, w, h, brightness)

Zeichnet ein Rechteck an der Koordinate x,y mit der Breite w und der Höhe h. Beispiel:

```
drawRectangle(0,0,8,8);
```

oder

```
drawRectangle(0,0,8,8,100);
```

drawTriangle(x0, y0, x1, y1, x2, y2, brightness)

Zeichnet ein Dreieck. Du teilst hier die Koordinaten der drei Ecken mit.

fillDisplay(brightness)

Alle Pixel leuchten in der maximalen bzw. definierten Helligkeit. Beispiele:

```
fillDisplay();
```

oder

```
fillDisplay(100);
```

findIBeacon(beaconName)

Liefert ein true, wenn die gesuchte Karte (identifiziert mit einer Nummer oder Text) in der Nähe ist. Falls die Karte nicht in der Nähe ist, wird entsprechend ein false geliefert.

getOrientation()

Liefert einen Wert zwischen 1 und 4, je nach Orientierung der Karte.

```
byte o = getOrientation();
```

getTimerSeconds();

Liefert dir als Rückgabewert die Anzahl der vergangenen Sekunden seit dem Start der Karte.

```
int sekunden = getTimerSeconds();
```

siehe auch `resetTimer()`

getXAcceleration()

Liefert einen positiven Wert, wenn eine Beschleunigung um die X-Achse im Uhrzeigersinn festgestellt wird. bzw. einen Negativen im Gegenuhrzeigersinn.

```
int x = getXAcceleration();
```

getYAcceleration()

Identisch mit getXAcceleration, jedoch für die Y-Achse.

getZAcceleration()

Identisch mit getXAcceleration, jedoch für die Z-Achse.

handleAutoTurnOff(seconds)

Diese Anweisung stellt den Computer nach 120 Sekunden ab. Du kannst die Zahl ändern, um das Gerät früher oder später in den Schlafmodus zu versetzen.

isLeftButtonPressed()

Liefert true, wenn der linke Button gedrückt wird. Du kannst diese Funktion in einem if-Block brauchen.

isMiddleButtonPressed()

Siehe isLeftButtonPressed, jedoch für den mittleren Button.

isOrientationDown()

Gleich wie isOrientationUp, jedoch wird diese Funktion true, wenn du die Karte umgekehrt auf den Tisch legst.

isOrientationHorizontally()

Gleich wie isOrientationUp, jedoch wird diese Funktion true, wenn du die Karte hochkant hältst.

isOrientationUp()

Liefert ein true, wenn du die Karte mit dem Bildschirm gegen oben auf den Tisch legst. Du kannst diesen Befehl in einer if-Anweisung brauchen:

```
If(isOrientationUp()) {  
    ...  
}
```

isOrientationVertically()

Gleich wie isOrientationUp, jedoch wird diese Funktion true, wenn du die Karte quer hältst.

isRightButtonPressed()

Siehe isLeftButtonPressed, jedoch für den rechten Button.

noTone()

Beendet den aktuellen Ton.

playMelody(melody, durations, count)

Spielt eine Melodie ab. Die Töne und die Dauer der Töne müssen in Form eines Arrays übergeben werden. Beispiel in Kapitel „Spiel mir ein Lied“ auf Seite 88.

print(textOrNumber)

Druckt eine Zahl oder einen Text im Fenster des seriellen Monitors aus. Das wird häufig verwendet zur Fehlersuche.

```
print(„ich bin hier“);
```

```
byte einWert = 100;  
print (einWert);
```

println(textOrNumber)

Dasselbe wie print, jedoch erfolgt danach noch ein Zeilenumbruch (println = print line).

resetOx0card()

Löscht den momentanen Zustand und startet die Ox0card neu.

resetTimer

Diese Funktion stellt den internen Sekundenzähler auf 0. Siehe auch getTimerSeconds().

setupAsIBeacon(beaconNr)

Diese Anweisung macht aus unserer Karte einen virtuellen Leuchtturm und schickt hierbei eine Nummer oder Text als Funksignal.

tone(frequency, duration)

Spielt einen Ton in der angegebenen Frequenz ab. Wenn die Dauer (duration) nicht angegeben wird, bleibt der Ton bestehen, bis „noTone()“ aufgerufen wird. Die Dauer wird in Millisekunden angegeben:

```
tone(440);  
delay(1000);  
noTone();
```

oder

```
tone(440,1000);
```

Du kannst auch die Symbole aus der Tonleiter brauchen:

```
tone(NOTE_A4,1000);
```

turnDisplayOn();

Alle Pixel leuchten in der maximalen Helligkeit.

turnOff() => turnOff(leftButton, middleButton, rightButton)

Stellt das Gerät aus (Standby) und mittels den Parametern kann zusätzlich bestimmt werden, welche Buttons die Ox0card wieder starten.

turnOff();

Stellt das Gerät aus (Standby).

KONSTANTEN DER TONLEITER



Die folgende Tabelle enthält alle Tonnamen, die du bei den Funktionen `tone()` und `playMelody` verwenden kannst:

Die Namen entsprechen jeweils den englischen Bezeichnungen der Tonleiter; die Zahl gibt die Oktave an.

KONSTANTE	FREQUENZ	KONSTANTE	FREQUENZ
NOTE_B0	31	NOTE_FS2	93
NOTE_C1	33	NOTE_G2	98
NOTE_CS1	35	NOTE_GS2	104
NOTE_D1	37	NOTE_A2	110
NOTE_DS1	39	NOTE_AS2	117
NOTE_E1	41	NOTE_B2	123
NOTE_F1	44	NOTE_C3	131
NOTE_FS1	46	NOTE_CS3	139
NOTE_G1	49	NOTE_D3	147
NOTE_GS1	52	NOTE_DS3	156
NOTE_A1	55	NOTE_E3	165
NOTE_AS1	58	NOTE_F3	175
NOTE_B1	62	NOTE_FS3	185
NOTE_C2	65	NOTE_G3	196
NOTE_CS2	69	NOTE_GS3	208
NOTE_D2	73	NOTE_A3	220
NOTE_DS2	78	NOTE_AS3	233
NOTE_E2	82	NOTE_B3	247
NOTE_F2	87	NOTE_C4	262

KONSTANTE	FREQUENZ	KONSTANTE	FREQUENZ
NOTE_CS4	277	NOTE_DS6	1245
NOTE_D4	294	NOTE_E6	1319
NOTE_DS4	311	NOTE_F6	1397
NOTE_E4	330	NOTE_FS6	1480
NOTE_F4	349	NOTE_G6	1568
NOTE_FS4	370	NOTE_GS6	1661
NOTE_G4	392	NOTE_A6	1760
NOTE_GS4	415	NOTE_AS6	1865
NOTE_A4	440	NOTE_B6	1976
NOTE_AS4	466	NOTE_C7	2093
NOTE_B4	494	NOTE_CS7	2217
NOTE_C5	523	NOTE_D7	2349
NOTE_CS5	554	NOTE_DS7	2489
NOTE_D5	587	NOTE_E7	2637
NOTE_DS5	622	NOTE_F7	2794
NOTE_E5	659	NOTE_FS7	2960
NOTE_F5	698	NOTE_G7	3136
NOTE_FS5	740	NOTE_GS7	3322
NOTE_G5	784	NOTE_A7	3520
NOTE_GS5	831	NOTE_AS7	3729
NOTE_A5	880	NOTE_B7	3951
NOTE_AS5	932	NOTE_C8	4186
NOTE_B5	988	NOTE_CS8	4435
NOTE_C6	1047	NOTE_D8	4699
NOTE_CS6	1109	NOTE_DS8	4978
NOTE_D6	1175		

LÖSUNGEN



- **Wecker**
- **Induktionsherd**
- **Heizung**
- **Radio**
- **Handy**
- **Taschenrechner**
- **Schulglocke**
- **Kaffeemaschine**
- **Ofen**
- **Velolicht**
- **Auto**
- **TV**
- **Tablet**

Kapazitiver Touch-Sensor

Erkennt die Position der Finger auf dem Glas.

Distanz-Sensor

Erkennt, ob du Dein Handy am Ohr hast und schaltet den Touch-Screen und den Bildschirm aus.

Mikrofon

Nimmt Umgebungsgeräusche und Stimmen auf. Mittels Software werden störende Umgebungsgeräusche beim Telefonieren rausgefiltert.

Kamera

Für coole Selfies und Video-Clips.

Licht-Sensor

Misst das Umgebungslicht; je mehr, desto heller leuchtet der Bildschirm.

Beschleunigungssensor

Misst, wenn du Dich im Raum bewegst.

Gyro-Sensor

Misst den Winkel im dreidimensionalen Raum (statisch).

GPS-Antenne

Empfängt die Signale der GPS-Sender und berechnet mittels Triangulation, wo du momentan gerade bist.

GSM-Antenne

Empfängt Daten- und Sprachnachrichten deines Telekomanbieters.

WiFi-Antenne

Empfängt Daten aus dem WiFi-Netzwerk.

Bluetooth-Antenne

Empfängt Daten deines drahtlosen Kopfhörers.

Biege / Drucksensor

Beim iPhone kannst du den Button unterschiedlich stark drücken. Das wird mit einem Drucksensor erkannt.

NFC-Antenne

Damit kannst du die Nummer einer Kreditkarte lesen.

Taster

Mit den Tastern kannst du Dinge ein/ ausschalten, die Lautstärke regeln etc. Jeder Taster ist ein sogenannter haptischer Sensor.

Batteriespannungssensor

Er misst, wie stark die Batterie entladen ist.

Transistor

Er erkennt, ob das Kopfhörerkabel eingesteckt ist.

Transistor

Er erkennt, ob das Gerät an einem Computer oder einer Ladestation angeschlossen ist.

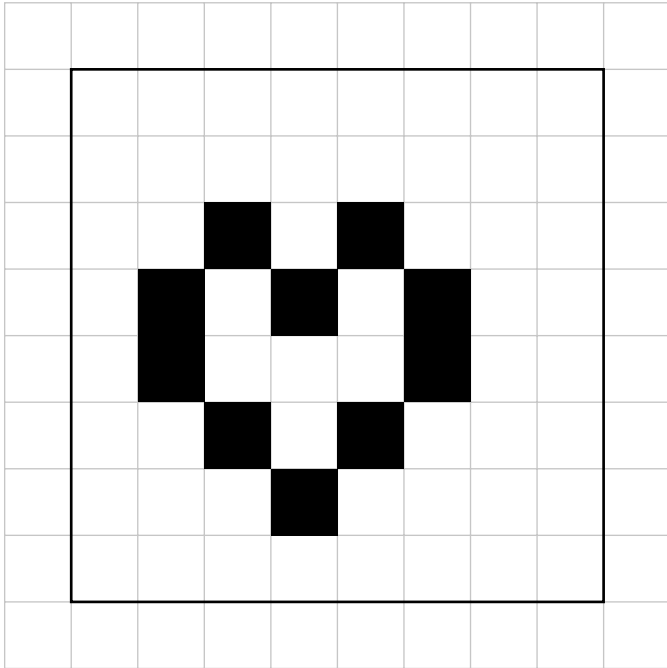
Drucksensor

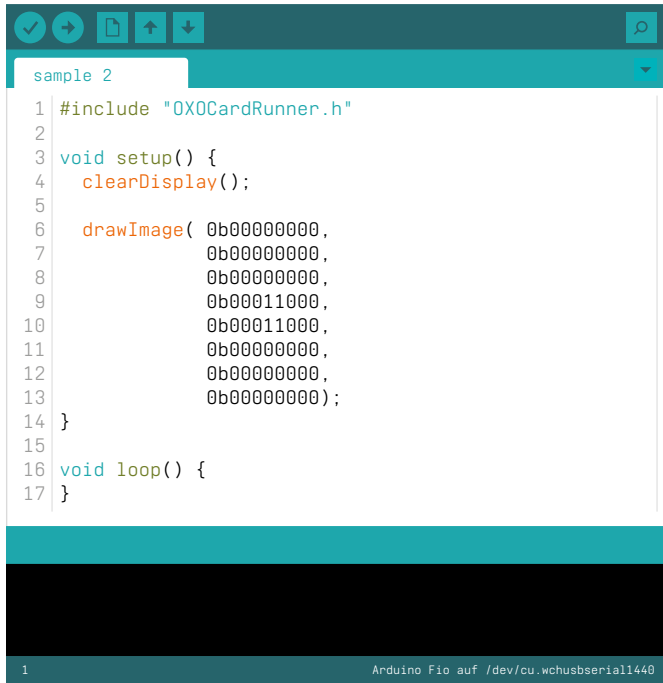
Dieser arbeitet auch im Hintergrund und misst den Luftdruck. Er wird verwendet, um das GPS-Signal zu verbessern.

Temperatur-Sensor

Dieser Sensor wird dafür verwendet, andere, temperatur-empfindliche Sensoren zu kalibrieren. Temperatur hat zum Beispiel einen Einfluss auf den Drucksensor und muss eingerechnet werden.

Wie gross war deine Liste? Es ist schon erstaunlich, was die Entwickler in so ein kleines Gerät reinpacken. Erstaunlich ist auch, dass dieses Gerät über mehr Sinne verfügt als ein Mensch!






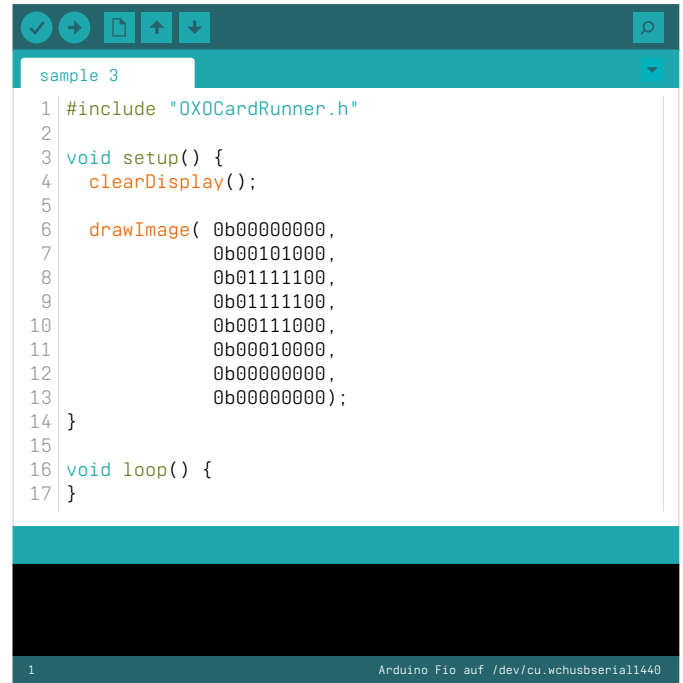
```

1 #include "OX0CardRunner.h"
2
3 void setup() {
4   clearDisplay();
5
6   drawImage( 0b00000000,
7              0b00000000,
8              0b00000000,
9              0b00011000,
10             0b00011000,
11             0b00000000,
12             0b00000000,
13             0b00000000);
14 }
15
16 void loop() {
17 }

```

1 Arduino Fio auf /dev/cu.wchusbserial11440

 Datei [sample2](#).




```

1 #include "OX0CardRunner.h"
2
3 void setup() {
4   clearDisplay();
5
6   drawImage( 0b00000000,
7              0b00101000,
8              0b01111100,
9              0b01111100,
10             0b00111000,
11             0b00010000,
12             0b00000000,
13             0b00000000);
14 }
15
16 void loop() {
17 }

```

1 Arduino Fio auf /dev/cu.wchusbserial11440

Links siehst du die Original-Datei und rechts wie der Code aussehen sollte.

 Datei [sample3](#).

```

sample 4
1 #include "OX0CardRunner.h"
2
3 void setup() {
4   clearDisplay();
5 }
6
7 void loop() {
8   clearDisplay();
9   drawImage( 0b00000000,
10             0b00101000,
11             0b00101000,
12             0b00000000,
13             0b01000100,
14             0b00111000,
15             0b00000000,
16             0b00000000);
17   delay(1000);
18
19   clearDisplay();
20   drawImage( 0b00000000,
21             0b00101100,
22             0b00100000,
23             0b00000000,
24             0b00000100,
25             0b00111000,
26             0b00000000,
27             0b00000000);
28   delay(1000);
29 }

```

↑ Datei [sample4.](#)

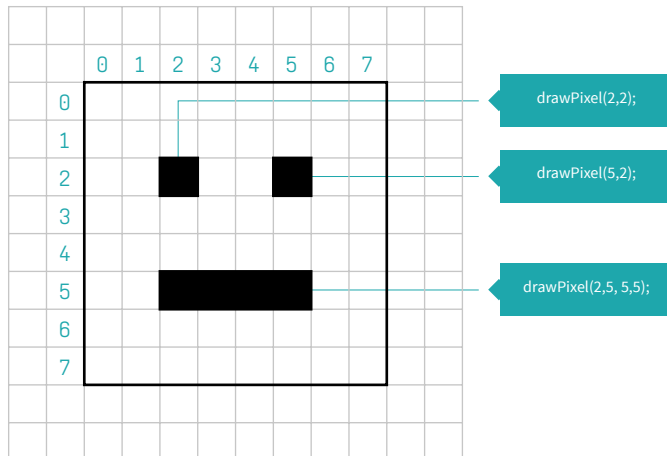
```

sample 5
1 #include "OX0CardRunner.h"
2
3 void setup() {
4   clearDisplay();
5 }
6
7 void loop() {
8   drawImage( 0b00000000,
9             0b00101000,
10            0b00101000,
11            0b00000000,
12            0b01000100,
13            0b00111000,
14            0b00000000,
15            0b00000000);
16
17   if (isRightButtonPressed()) {
18     clearDisplay();
19     drawImage( 0b00000000,
20             0b00101100,
21             0b00100000,
22             0b00000000,
23             0b00000100,
24             0b00111000,
25             0b00000000,
26             0b00000000);
27     delay(1000);
28     clearDisplay();
29   }
30 }


```

↑ Datei [sample5.](#)

Spätestens jetzt kannst du deine Kreativität vollständig ausleben, daher verzichten wir auf eine vollständige Beispiellösung. Als Inspiration soll folgende Skizze dienen:


**Zusatz:**

Besprich die verschiedenen Lösungen mit deinen Kollegen. Wer hat das coolste Smiley, wer hat dafür am wenigsten Zeilen Code gebraucht? Bist du der Coding-Hero? 😊



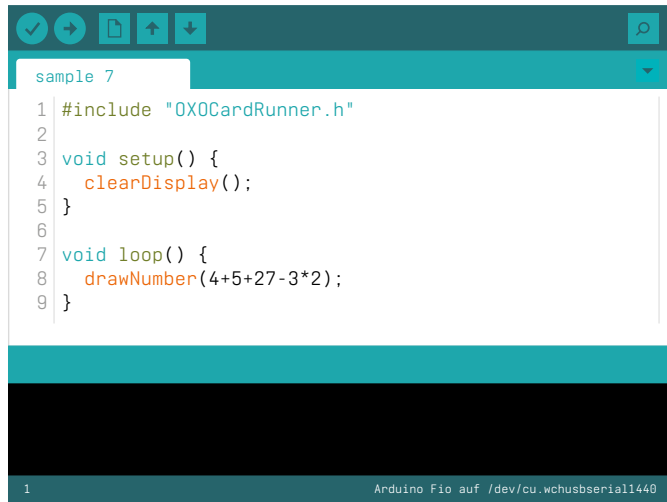
```
1 #include "0X0CardRunner.h"
2
3 void setup() {
4     clearDisplay();
5 }
6
7 void loop() {
8
9     drawCircle(1,1,1,20);
10    drawRectangle(4,0,3,3,50);
11    drawPixel(1,5,255);
12    drawTriangle(7,3,7,7,2,7,100);
13
14    delay(1000);
15
16    drawRectangle(4,0,3,3,0);
17    drawPixel(1,5,0);
18
19    delay(1000);
20
21 }
```

1 Arduino Fio auf /dev/cu.wchusbserial1440

 Datei [sample6c.](#)

Man kann diese Aufgabe auf zwei Arten lösen. Zum einen mit zwei Pixelbildern, die du einfach nacheinander anzeigst oder eben so, wie das im `sample6c` gezeigt wird, mit den Grafikbefehlen.

Wir ersetzen bei diesem Beispiel nur noch die Teile, die sich zwischen den beiden Bildern geändert haben. Bereits auf unserem kleinen Bildschirm mit nur 64 Pixels macht dies einen Unterschied aus. Da der Computer beim zweiten Fall weniger Pixel ändern muss, läuft die Animation flüssiger und flackert weniger.



```
1 #include "OXOCardRunner.h"
2
3 void setup() {
4   clearDisplay();
5 }
6
7 void loop() {
8   drawNumber(4+5+27-3*2);
9 }
```

1 Arduino Fio auf /dev/cu.wchusbserial1440

Hast du die Punkt-vor-Strich-Regel beachtet? Dann solltest Du auf dasselbe Resultat gekommen sein, wie der Computer.

↑ Datei `sample7.`

Wenn du mit Variablen rechnest, ist es genau gleich, wie wenn dort Zahlen stehen würden. Die Punkt-vor-Strich-Regel gilt gleich. Hast du dasselbe bekommen?


↑ Datei `sample7b.`

Dieses Programm hat eine Variable `a`, welche mit `drawNumber` ausgegeben wird. Danach wird etwas erwartet – das kennst du ja bereits – und gleich anschliessend haben wir wieder einen `if`-Block, den wir bei der Abfrage der Buttons schon gebraucht haben. Der `if`-Block kann noch eine ganze Menge mehr, als wir bisher gesehen haben. Man kann beispielsweise damit auch Variablen-Werte abfragen und je nach Vergleich Blöcke von Befehlen ausführen lassen.

 Datei `sample7c.`

Ändere in der Bedingung einfach „`a < 8`“ in „`a < 50`“ ab

Ändere die Anweisung „`a = a + 1`“ in „`a = a + 5`“ ab.

 Datei `sample7c.`

a.)
Ändere die Zeile `delay(50)` z.B. in `delay(25)` oder `delay(100)`
ab. Bei 25 wird die Animation doppelt so schnell, bei 100 doppelt so langsam.

b.)
Du kannst beliebig viele Pixel runterfallen lassen, wenn Du weitere „drawPixel“- Anweisungen hinzufügst, wie das folgende Beispiel zeigt:

```
drawPixel(3,a);
drawPixel(5,a);
delay(50);
drawPixel(3,a,0);
drawPixel(5,a);
```

c.)
Ändere die Bedingung

```
if (a < 7) {
```

in folgende ab:

```
if (a < 4) {
```

Jetzt nimmt a nur noch die Werte 0,1,2,3 an und dein Pixel stoppt in der Mitte.



```
sample 8b
1 #include "0X0CardRunner.h"
2
3 void setup() {
4   clearDisplay();
5 }
6
7 void loop() {
8   byte r = random(4);
9
10  drawCircle(4,4,r);
11  delay(100);
12  drawCircle(4,4,r,0);
13 }
```

Wir erzeugen zuerst eine Variable, die wir r nennen und die mit random einen Zufallswert zwischen 0 und 4 erhält.

```
byte r = random(5);
```

Jetzt zeichnen wir den Kreis mit drawCircle, warten eine Weile und löschen den Kreis wieder:

```
drawCircle(4,4,r);
delay(100);
drawCircle(4,4,r,0);
```

 Datei `sample8b.`

Das Spektrum der Töne, die dein Gehör wahrnehmen kann, beginnen bei ca. 20 Hertz und enden je nach Alter bei ca. 20'000 Hertz, wo wir noch einen Ton wahrnehmen können. Tiere können teilweise höhere Frequenzen wahrnehmen.

Du hast vielleicht gemerkt, dass es nicht ganz einfach ist, einen bestimmten Ton einer Tonleiter zu treffen. Damit es präziser geht, haben wir für dich Symbole hinterlegt, die du anstelle der Frequenz angeben kannst. Beispielsweise kannst du folgendes anstelle des `tone(440)` schreiben:

```
tone(NOTE_A4);
```

Du findest die gesamte Liste der definierten Töne im Anhang „Konstanten der Tonleiter“. „A4“ ist die Bezeichnung des eingestrichenen A's in den USA.



Datei `sample9.`

Kopiere den Inhalt der Setup-Funktion in die Loop-Funktion, dann fängt das Programm immer von Neuem an. Um die Töne einzustellen, kannst du den Anfangswert `i=0` z.B. auf `i=500` stellen und bei der Bedingung `i < 4000` einen beliebigen anderen Wert angeben.

Zusatzinfos:

Es ist interessant zu sehen, wie sich ein Einzelton verändert, wenn man ihn in sehr kurzen Abständen ändert. Im obigen Beispiel haben wir innerhalb der Schleife keine zusätzliche Verzögerung eingebaut. Du kannst aber mal probieren, was geschieht, wenn du nach dem `tone()`-Aufruf einen sehr kurzen `delay()` einbaust, beispielsweise 5 oder 10 Millisekunden (z.B. `delay(5)`). Der Ton wird länger und du kannst die Einzeltöne besser unterscheiden. Mit diesen Möglichkeiten kannst du bereits eine Menge an Geräuschen erzeugen.



Datei `sample9b.`

Für die Tonleiter haben wir die Oktave A4 (a')
genommen. Du kannst aber natürlich auch eine andere
Oktave ausprobieren:

```
int melody[] = { NOTE_A4, NOTE_AS4, NOTE_B4,
                NOTE_C5, NOTE_CS5, NOTE_D5,
                NOTE_DS5, NOTE_E5, NOTE_F5,
                NOTE_FS5, NOTE_G5, NOTE_GS5};

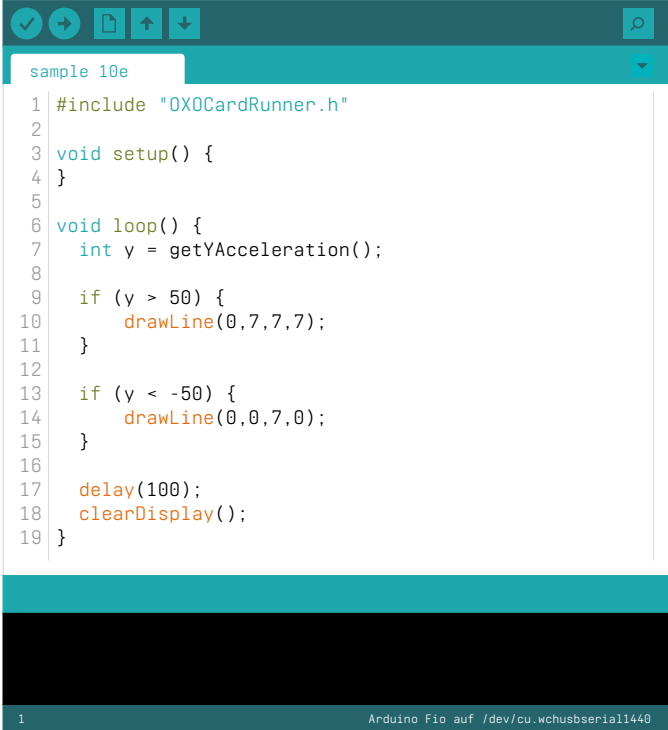
int durations[] = { 250,250,250,250,250,250,250,
                    250,250,250,250,250 };

playMelody(melody,durations,8);
```

```
sample 10b
1 #include "OX0CardRunner.h"
2
3 void setup() {
4 }
5
6 void loop() {
7   byte o = getOrientation();
8   if (o == 1) {
9     drawImage( 0b00000000,
10              0b01101100,
11              0b00000000,
12              0b00000000,
13              0b01000100,
14              0b00111000,
15              0b00000000,
16              0b00000000);
17 }
18 else if (o == 4) {
19   drawImage( 0b00000000,
20            0b11101110,
21            0b10101010,
22            0b00000000,
23            0b01000100,
24            0b00111000,
25            0b00000000,
26            0b00000000);
27 } else {
28   clearDisplay();
29 }
30 }
```

↑ Datei `sample10b`.

Bist du auf die gleiche Lösung gekommen? Es gibt, wie schon an anderer Stelle gesagt, immer mehrere Wege, eine Aufgabe zu lösen. In unserem Beispiel prüfen wir, ob die Karte liegt. Falls dies nicht der Fall ist, prüfen wir weiter (`else`-Fall), ob die Karte hochkant steht. Wenn beides nicht eintrifft, löschen wir den Bildschirm. Wenn du das ohne „`else`“ gemacht hast, kann es möglicherweise sein, dass sich das Programm nicht genau so verhält, wie du es erwartest. Solche falsch programmierten Abläufe sind sehr häufig in der Programmierung und manchmal ist es unglaublich aufwändig, einen solchen Fehler zu beheben. Falls es nicht geklappt hat, findest du die Lösung in [sample10b](#).



```
1 #include "0X0CardRunner.h"
2
3 void setup() {
4 }
5
6 void loop() {
7   int y = getYAcceleration();
8
9   if (y > 50) {
10      drawLine(0,7,7,7);
11   }
12
13   if (y < -50) {
14      drawLine(0,0,7,0);
15   }
16
17   delay(100);
18   clearDisplay();
19 }
```

 Datei [sample10e](#).

IMPRESSUM

VERSION 1.02

Alle Angaben in diesem Buch sind urheberrechtlich geschützt und dürfen in keiner Form ohne schriftliche Bestätigung der OXON AG weitergegeben oder vervielfältigt werden.

Entwickelt im Liebefeld
(Schweiz) durch:

OXON AG

Waldeggstrasse 47
CH-3097 Liebefeld
Schweiz
www.oxon.ch
info@oxon.ch

Projektteam:

Thomas Garaio (Buch, Konzept, Design)
Jascha Haldemann / Tobias Meerstetter (Elektronik, Firmware)
René Rügsegger (Grafik-Design)
Aveline Kompis (Didaktik, Lektorat)

Erste Ausgabe, April 2017.

DESIGNED  
IN-LIEBEFELD

HANDMADE
THING 

 MADE WITH PASSION



WWW.OXOCARD.CH

MADE
BY **OXON**[®]