Arduino, Elektrizität und Programmieren: Praxis

In den vorangegangenen Arbeitspaketen werden grundlegende Vorstellungen zur Elektrizität dargeboten, die Realisierung erster eigener Schaltungen mit Steckbrett und Arduino-Board angeleitet und das Konzept der Variablen zum Speichern von Werten samt elementaren Strukturen zur Organisation von Programmcode erklärt und eingeübt. Zudem wird anhand des Morsealphabets die Idee der Codierung nähergebracht. Diese drei Aspekte werden in diesem Arbeitspaket wiederholt, vertieft und erweitert. In den vier Arbeitsblättern werden neue Bauteile für Schaltungen verwendet, weitere Programmierkonzepte kennen gelernt sowie umgesetzt und mit der in der Musik üblichen Notenschrift wird ein weiteres Beispiel für Codierung, nämlich für die Codierung von Tönen, vorgestellt:

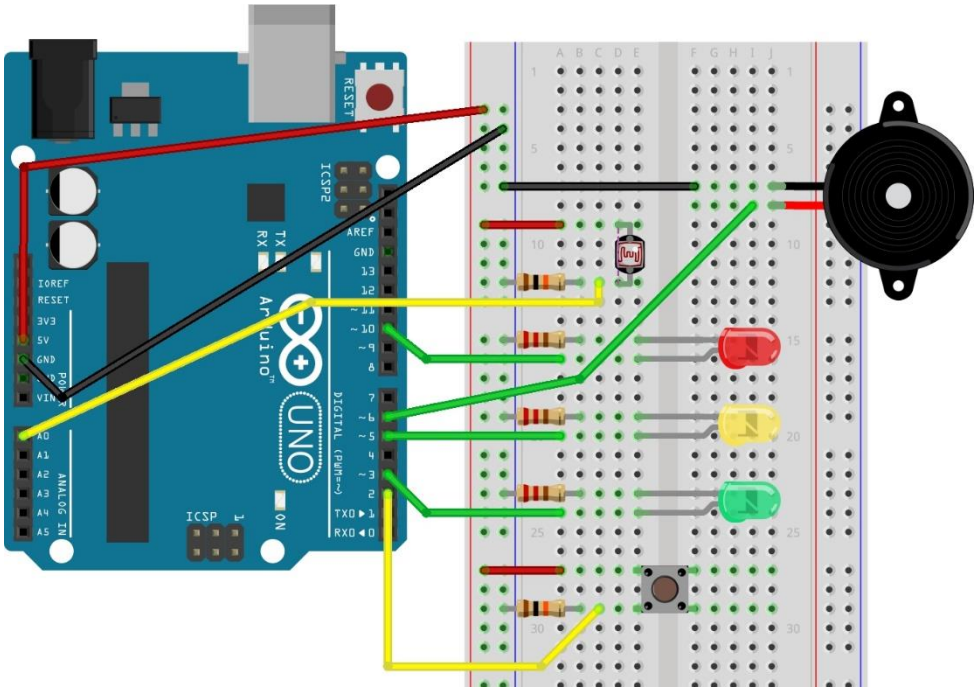
- Im ersten der vier Informations- bzw. Arbeitsblätter wird ein **Piezo-Lautsprecher** verwendet, um programmgesteuert Töne zu erzeugen. Zur Kombination von Tonfolgen zu Melodien wird das Notations- und Codierungskonzept der Notenschrift wiederholt bzw. vorgestellt und damit im Zusammenhang das informatische Konzept der Feldvariable (engl.: des »Arrays«) zum Speichern mehrerer Werte desselben Datentyps erklärt und eingeübt. Solcherart können einfache Lieder codiert und mit Hilfe des Arduino-Boards „abgespielt“ werden.
- Die Verwendung von **Feldvariablen (Arrays)** steht im Mittelpunkt des zweiten Informations- bzw. Arbeitsblattes. Neben der Dateneingabe in Feldvariablen mit Hilfe von Programmschleifen und der Ausgabe von Feldwerten über den seriellen Monitor, einem Zusatzprogramm zur Arduino-Programmierumgebung, wird gezeigt, wie mit selbst programmierten Befehls- und den neuen Frage-Programmblöcken Operationen mit Werten von Feldvariablen modularisiert werden können. Besonderes Augenmerk wird darüber hinaus auf die Analyse von Berechnungen mit Werten aus Feldvariablen gelegt. Dazu wird einerseits das bereits aus dem zweiten Arbeitspaket bekannte mentale Modell zur Versinnbildlichung von Variablen und deren Inhalten als „rechteckige Speicherzellen“ verwendet, andererseits wird eine alternative effizientere Darstellungsform mit Tabellierung der Variablenwerte erklärt und eingeübt.
- Auch das dritte Informations- bzw. Arbeitsblatt widmet sich der Anwendung von Feldvariablen (Arrays). Als geeigneter Speicherort für **Messwerte** von Sensoren motiviert die Datenstruktur Feldvariable (Array) zur Auseinandersetzung mit dieser automatisierten Form der Datenerhebung und einfacher statistischer Verfahren der Datenauswertung. Neben dem bereits bekannten Photowiderstand werden ein Temperatursensor und ein Ultraschallsensor für die Datenerhebung verwendet. Die Berechnung statistischer Maßzahlen aus den erhobenen Rohdaten nutzt wieder die im vorherigen Arbeitsblatt kennen gelernten Frage-Unterprogramme und motiviert die Zusammenfassung mehrerer „zusammengehöriger“ Unterprogramme in sogenannte Programmbibliotheken (engl.: Libraries). Die Codierung eigener Programmbibliotheken (Libraries) und deren Einbindung in eigene Arduino-Programme rundet die Beschäftigung mit Feldvariablen (Arrays) ab.
- Das vierte Informations- und Arbeitsblatt nutzt die zuvor erworbenen Fertigkeiten, um die Vorstellungen über Elektrizität zu vertiefen. Aufbauend auf den Energietransport im geschlossenen elektrischen Stromkreis wird über die Analogie des Drucks in Flüssigkeiten und Gasen ein tragfähiges mentales Modell der elektrischen **Spannung** und in weiterer Folge des Ohm’schen

Widerstandes entwickelt. Neben der Messung von Widerstand und Spannung stellt die Anwendung Ohm'scher Widerstände zur Spannungsteilung im Mittelpunkt der Programmierung des Arduino-Boards zur Vertiefung dieser ansonsten recht theoretischen Konzepte. Dabei werden neben dem seriellen Monitor oder dem Piezo-Lautsprecher auch neue elektronische Bauteile wie Drehpotentiometer oder Flüssigkristallanzeige verwendet, um elektrische Größen auch sinnlich wahrnehmbar zu machen.

Wie gewohnt bauen die vier Arbeitsblätter samt zugehörigen Informationsdateien aufeinander auf und sollten wieder in der Reihenfolge ihrer Nummerierung (**ST_I09…** bis **ST_I12** bzw. **ST_AA09**… bis **ST_AA12**) bearbeitet werden.

Programmierpraxis mit Schall

Für die Experimente zur Schallerzeugung mit dem Arduino-Board kann die bereits bekannte Schaltung zum Programmieren einer Ampelsteuerung bzw. eines optischen Morseapparats weiterverwendet werden. Die notwendige Erweiterung der Schaltung besteht im Hinzufügen eines Piezo-Lautsprechers z.B. gemäß der nebenstehenden Abbildung :



Die Programmierung dieses Lautsprechers erfolgt mit den bekannten Befehlen **digitalWrite**, und **delay**, mit denen dem Lautsprecher durch Ein- und wieder Ausschalten ein „Knacken“ entlockt werden kann, oder mit den Befehlen **tone**, **delay**, und **notone**, mit denen ein Ton bestimmter Frequenz (entsprechend einer bestimmten Tonhöhe) erzeugt werden kann. Die Verwendung dieser Befehle kann den – auch in der Informationsdatei **ST_I09Arduino_Praxis_Schall** abgebildeten – Codefragmenten entnommen werden:

```
int piezoPin;

void setup() {
  piezoPin = 6;
  pinMode(piezoPin,OUTPUT);
}

void loop() {
  digitalWrite(piezoPin, HIGH);
  delay(50);
  digitalWrite(piezoPin, LOW);
  delay(50);
}
```

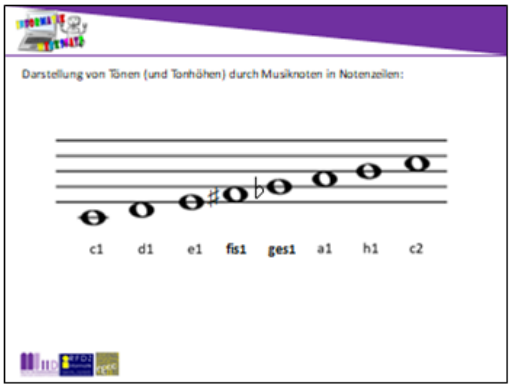
```
int piezoPin;

void setup() {
  piezoPin = 6;
  pinMode(piezoPin,OUTPUT);
}

void loop() {
  tone(piezoPin,440);
  delay(2000);
  noTone(piezoPin);
  tone(piezoPin,396);
  delay(1000);
  noTone(piezoPin);
}
```

Die Zuordnung der Frequenzen zu den eventuell aus dem Musikunterricht bekannten Noten ist der nachfolgenden Tabelle zu entnehmen:

Ton:	c	cis/des	d	dis/es	e	f	fis/ges	g	gis/as	a	ais/b	h
Frequenz:	264	278,4	297	316,8	330	352	371,3	396	422,4	440	475,2	495

Dabei werden in der Musik die Noten durch die Positionierung der Notensymbole entsprechend ihrer Tonhöhe in fünf Notenlinien codiert. Da die Kenntnis diese Form der Codierung bzw. der Decodierung der Töne aus der „Notenschrift“ nicht vorausgesetzt wird, wird sie der Informationsdatei **ST_I_09Arduino_Praxis_ Schall** anhand von Abbildungen grundlegend erklärt. Die Abbildungen (inklusive Textvorschlägen für die Erklärung) finden sich auch in der Kurzpräsentation **ST_PR_09Arduino_Praxis_ Schall**:

Inhaltsfolie 1 zur Codierung von Tönen in Notenlinien:



Textvorschlag:

In der Musik werden Töne durch Musiknoten in Notenzeilen dargestellt – die Position der Note in den fünf Notenzeilen gibt Auskunft über die Tonhöhe. Die Folie zeigt die sieben eingestrichenen Stammtöne samt dem zweigestrichenen c.

Die restlichen Noten aus der Tabelle auf der vorangehenden Folie entstehen durch Erhöhung um einen Halbton mit dem Kreuz-Versetzungszeichen, ♯ (Einblendung durch Drücken der Enter-Taste), bzw. durch Erniedrigung um einen Halbton mit dem b-Versetzungszeichen, ♭ (Einblendung durch Drücken der Enter-Taste).

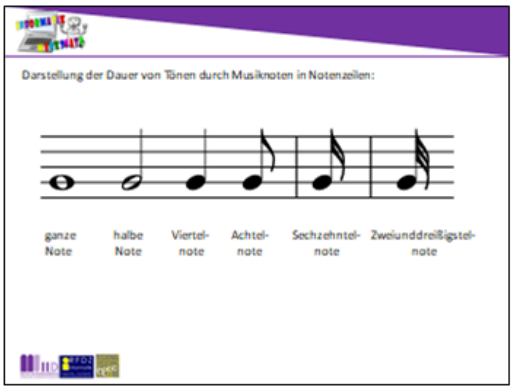
Die mit dem Kreuz-Versetzungszeichen erhöhten Noten heißen dann:

cis, dis, fis, gis, ais,

die mit dem b-Versetzungszeichen erniedrigten Noten heißen:

des, es, ges, as, b (statt hes)

Inhaltsfolie 2 zur Codierung von Tönen in Notenlinien:



Textvorschlag:

Wie lange ein Ton erklingen soll, wird in der Musik durch unterschiedliche Notenformen ausgedrückt. Auf der Folie sind diese Notenformen mit den zugehörigen Notenwerten dargestellt.

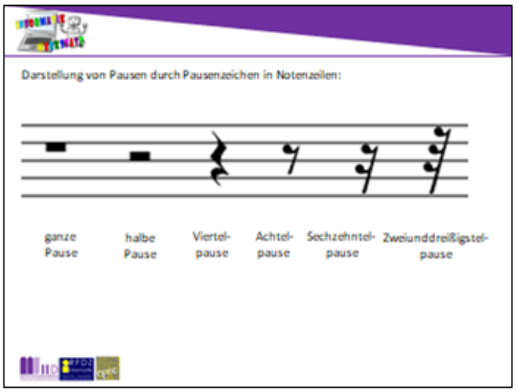
Dabei gilt:

Ein Ton, der durch eine halbe Note dargestellt wird, dauert halb so lang wie einer, der durch eine ganze Note dargestellt wird.

Ein Ton, der durch eine Viertelnote dargestellt wird, dauert halb so lange wie einer, der durch eine halbe Note dargestellt wird.

...usw.

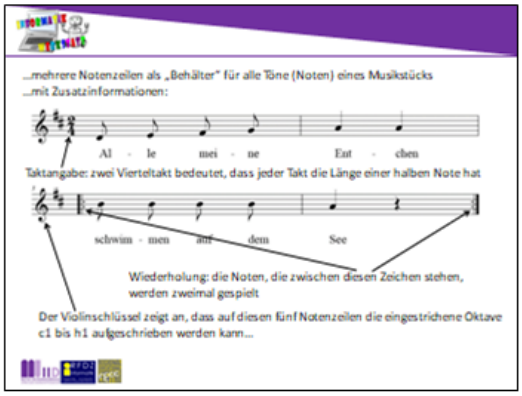
Inhaltsfolie 3 zur Codierung von Tönen in Notenlinien:



Textvorschlag:

In der Musik treten neben Tönen auch Pausen auf, die mit eigenen Pausenzeichen in den Notenlinien notiert werden. Aus der Folie ist ersichtlich, dass die Pausenlängen dabei genau den Notenlängen entsprechen.

Inhaltsfolie 4 zur Codierung von Tönen in Notenlinien:



Textvorschlag:

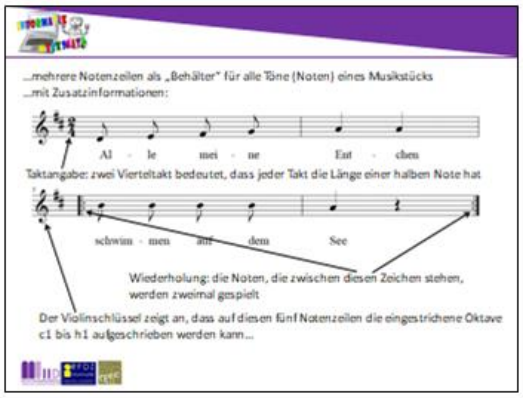
Ein großer Vorteil von Notenzeilen ist, dass mit ihnen ALLE Töne eines Musikstück zusammen aufgeschrieben werden können. Auf der Folie ist der Anfang des bekannten Liedes „Alle meine Entchen" mit Noten- und Pausenzeichen in Notenzeilen aufgeschrieben. Dabei werden noch folgende „Zusatzinformationen" angezeigt:

- die Taktlänge (Einblendung nach Drücken der Enter-Taste): Vorlesen des eingeblendeten Textes, Zeigen der Takt-Trennlinien;
- die Wiederholungszeichen (Einblendung nach Drücken der Enter-Taste): Vorlesen des eingeblendeten Textes (allenfalls „Nachrechnen" der richtigen Taktlänge am Beispiel eines Taktes (z.B. 4. Takt: Eine Viertelnote und eine Viertelpause haben die Länge von zwei Viertel...)
- der Violinschlüssel (Einblendung nach Drücken der Enter-Taste): Vorlesen des eingeblendeten Textes, allenfalls „Decodieren" der Dauer und Tonhöhe für die Notenzeichen.

Mit Hilfe der Notenschrift können in der Musik ganze Musikstücke in einem „Behälter" gespeichert werden. Die Größe eines solchen „Behälters" in der Musik reicht von einigen wenigen Notenzeilen bei kurzen Liedern bis zu vielen Notenblättern voll mit Notenzeilen bei umfangreicheren Kompositionen.

Der Wunsch, alle Töne eines Musikstücks zusammen speichern und dann mit Hilfe des `tone` -Befehls über den Piezo-Lautsprecher effizient „abspielen" zu können, motiviert die Einführung von Feldvariablen (engl. „Arrays"), die mehrere Werte ein und desselben Datentyps speichern können. Die Vereinbarung und die Verwendung derartiger Feldvariablen ist in der Informationsdatei **ST_I_09Arduino_Praxis_ Schall** und auch in der Kurzpräsentation **ST_PR_09Arduino_Praxis_Schall** erklärt:

Inhaltsfolie 1 zu „von Notenlinien zu Feldvariablen":



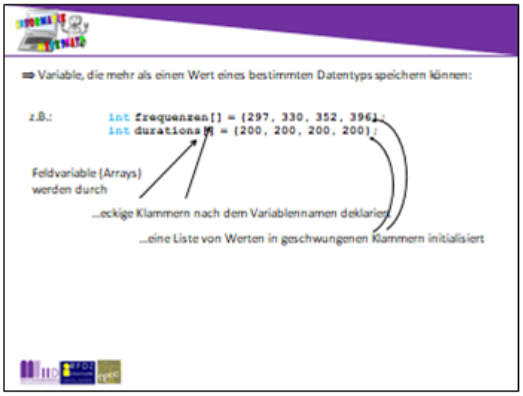
⇓

Textvorschlag:

Mit den bisherigen Kenntnissen zur Arduino-Programmierung muss jeder Ton EINZELN codiert werden – da sich von Ton zu Ton aber nur die Frequenz (und gegebenenfalls die Dauer) des Tones ändert (ändern), wäre es praktischer, diese Informationen – Frequenzen und Dauer der Töne – jeweils „in einem Behälter" zusammen speichern zu können....

Hinweis: Diese Folie (und insbesondere auch der vorgeschlagene Text) soll (sollen) die Einführung von Variablen motivieren, in denen mehrere Werte gleichzeitig gespeichert werden können (siehe nächste Folie).

Inhaltsfolie 2 zu „von Notenlinien zu Feldvariablen":



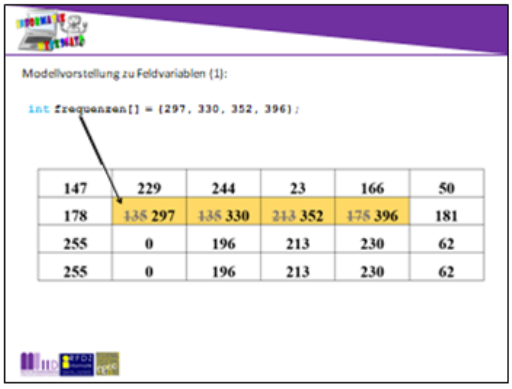
Textvorschlag:

Dafür bietet die Arduino-Programmierumgebung Feldvariable (auch: Arrays) an, in denen mehrere Werte von einem bestimmten Datentyp gespeichert werden können.

Die Deklaration einer Feldvariablen erfolgt durch das Setzen eines eckigen Klammerpaars nach dem Variablennamen (Einblendung),

die Initialisierung kann (z.B.) durch die Zuweisung einer Liste von Werten, die in geschwungene Klammern geschrieben werden muss, erfolgen (Einblendung)

Inhaltsfolie 3 zu „von Notenlinien zu Feldvariablen“:

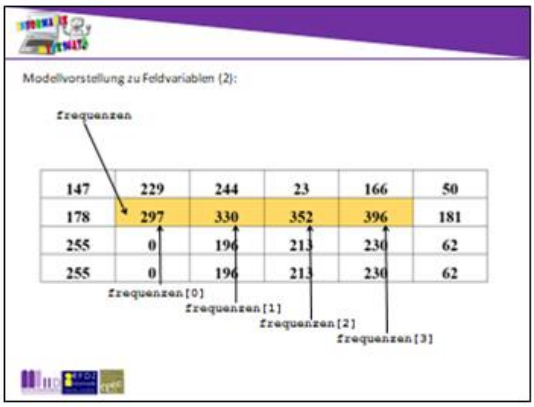


Textvorschlag:

Wird eine Feldvariable bei der Deklaration auch gleich initialisiert, dann werden so viele Speicherplätze reserviert, wie zum Speichern der als Liste übergebenen Werte notwendig sind – im gezeigten Beispiel also vier Speicherplätze für ganzzahlige Werte vom Datentyp `int`.

Zusätzlich werden durch die Initialisierung auch die Werte überschrieben, die zuvor in den reservierten Speicherplätzen gespeichert waren (Einblendung)

Inhaltsfolie 4 zu „von Notenlinien zu Feldvariablen“:



Textvorschlag:

Die Feldvariable verweist dabei auf den ersten der reservierten Speicherplätze, also auf jenen, der die niedrigste Adresse hat.

Um auf den Wert zuzugreifen, der in diesem ersten Speicherplatz gespeichert ist, muss man keinen Speicherplatz weitergehen. Im gezeigten Beispiel schreiben wir dafür:

`frequenzen [0]` (Einblendung)

Um auf den Wert zuzugreifen, der im zweiten Speicherplatz gespeichert ist, muss man einen Speicherplatz weitergehen. Im gezeigten Beispiel schreiben wir dafür:

`frequenzen [1]` (Einblendung)

Um auf den Wert zuzugreifen, der im drittenSpeicherplatz gespeichert ist, muss man zwei Speicherplätze weitergehen. Im gezeigten Beispiel schreiben wir dafür:

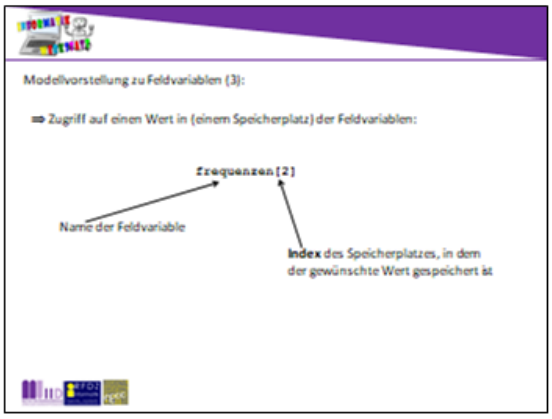
`frequenzen [2]` (Einblendung)

Um auf den Wert zuzugreifen, der im vierten Speicherplatz gespeichert ist, muss man drei Speicherplätze weitergehen. Im gezeigten Beispiel schreiben wir dafür:

`frequenzen [3]` (Einblendung)

Bemerkung: Auch wenn die Informationsdatei **ST_I_09Arduino_Praxis_Schall** bzw. das Arbeitsblatt **ST_AA_09Arduino_Praxis_Schall** zunächst vornehmlich auf die korrekte Verwendung von Feldvariablen (Arrays) abzielt, erscheint die Vermittlung eines mentalen Modells zu diesen „Variablen mit interner Struktur" von Beginn an unerlässlich. Dazu dienen insbesondere die obigen Inhaltsfolien 3 und 4, die an das mentale Modell von Variablen „ohne interne Struktur“ anknüpfen.

Inhaltsfolie 5 zu „von Notenlinien zu Feldvariablen“:



Textvorschlag:

Die Anzahl der Speicherplätze, die man vom ersten Speicherplatz der Feldvariable „weitergehen“ muss, um zum Speicherplatz des gewünschten Wertes zu gelangen, bezeichnet man als

Index

des Speicherplatzes.

Die Variable, mit der auf einen individuellen Wert in der Feldvariablen zugreifen kann, besteht daher aus

dem Namen der Feldvariablen

und

dem Index des gewünschten Speicherplatzes,

der in eckige Klammern geschrieben wird.

(am Beispiel `frequenzen[2]` auf der Folie zeigen)

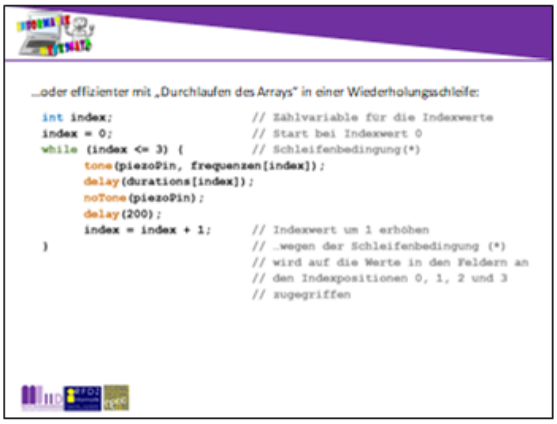
Inhaltsfolie 6 zu „von Notenlinien zu Feldvariablen“:

Textvorschlag:

Das „Merken“ aller Frequenzen und der Dauer aller Töne in Feldvariablen scheint zunächst noch keinen allzu großen Vorteil zu bringen, bleibt der Code doch offenbar gleich lang (Vergleich mit Folie 10).

...es fällt aber auf (auf Folie dabei auf die Indexwerte zeigen), dass auf die benötigten Werte durch **Hochzählen des Index** (hier kann auch darauf hingewiesen werden, dass der Index auch anders als durch Hochzählen verändert werden kann, z.B. kann er vom höchsten möglichen Wert „heruntergezählt“ werden usw.) zugegriffen werden kann...

Inhaltsfolie 7 zu „von Notenlinien zu Feldvariablen“:



Textvorschlag:

(in Fortsetzung von der vorhergehenden Folie)
Ein spürbarer Vorteil ergibt sich daher, wenn in einer Schleife auf die in der/den Feldvariablen gespeicherten Werte mit Hilfe einer Zählvariable zugegriffen wird...

Eine **Feldvariable** gibt einem **Bereich im Speicher des Computers** einen **Namen**, in dem **mehrere Daten desselben Datentyps** gespeichert werden können. Dieser Speicherbereich umfasst daher **mehrere elementare Speicherzellen**, auf die mit Hilfe einer **internen Adresse (beginnend bei 0** – sprich: Null) zugegriffen werden kann.

Das Programmieren des Piezo-Lautsprechers wird in mit vier Arbeitsanregungen der Datei **ST_AA_09Arduino_Praxis_Schall**“ eingeübt: Arbeitsanregungen **1)** und **2)** leiten eine Erweiterung des Codes zur „Ampelsteuerung“ bzw. eine Modifikation des Codes zum „Senden“ von Morse-Signalen an, während die letzten beiden Arbeitsanregungen sich „dem Musizieren“ widmen. Arbeitsanregung **4)** regt dabei zum „Jammen“ an: In Feldvariablen gespeicherten Töne eines Liedes sollen auf unterschiedliche Weise (z.B. „vom letzten zum ersten Ton“, „nur jeder zweite Ton“, …) „gespielt“ werden. Dadurch entstehen ungewohnte und überraschende Tonfolgen, die zu eigenem Weiterexperimentieren zum Durchlaufen der Werte von Feldvariablen (man sagt auch: **Traversieren von Feldvariablen**) motivieren können.

Arduino – Programmierpraxis mit Feldvariablen

Feldvariable können nicht nur „als Ganzes“ durch einmalige Zuweisung aller, in geschwungene Klammern eingeschlossenen Werte, „befüllt“ werden. Weitaus häufiger werden die Werte Index für Index in einer Schleife zugewiesen. Da bislang keine Möglichkeit zur direkten Benutzereingabe bekannt ist, beschränken sich die dargebotenen Beispiele und Erklärungen in der Informationsdatei **ST_I_10Arduino_Praxis_Feldvariable** auf die Zuweisung berechenbarer Werte an die einzelnen Speicherplätze von Feldvariablen.

Analog können die in einer Feldvariablen gespeicherten Werte mit einer Programmschleife über den Index auch ausgegeben werden. Für die Ausgabe wird zunächst der „serielle Monitor“, ein Zusatzprogramm der Arduino-Programmierumgebung genutzt.

Die Erklärungen der Informationsdatei sind wieder als Folien in der Kurzpräsentation **ST_PR_10_Arduino_Praxis_Feldvariable** verfügbar:

Inhaltsfolie 1 zu „Ein- und Ausgabe von Feldvariablen“:

Textvorschlag:

Feldvariable sind schon vom Abschnitt über die Programmierung des Piezo-Lausprechers bekannt, wo sie zum „Merken“ von Melodien verwendet wurden.

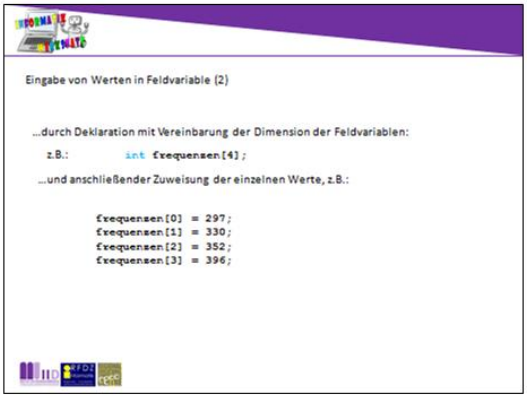
Dort wurden die Feldvariablen bei der Deklaration durch Zuweisung einer Werteliste mit Anfangswerten belegt (man sagt auch: Initialisiert).

Werte in einer Werteliste werden von geschwungenen Klammern umschlossen (zeigen)

Die Größe der Feldvariable wird in diesem Fall automatisch durch die Anzahl der Werte in der Werteliste festgelegt.

Statt „Größe der Feldvariablen“ sagt man auch „Dimension der Feldvariablen“.

Inhaltsfolie 2 zu „Ein- und Ausgabe von Feldvariablen“:



Textvorschlag:

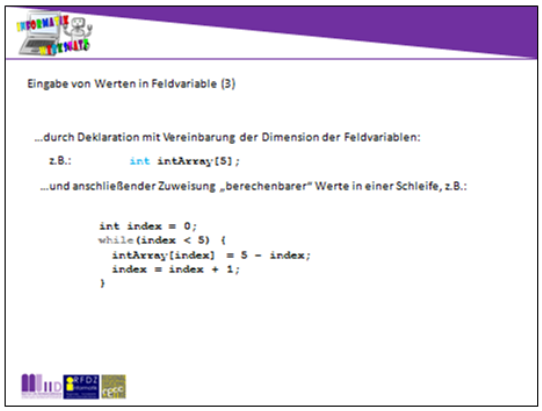
...häufiger wird aber die Dimension der Feldvariablen bei der Deklaration festgelegt – die Dimension wird dabei in eckigen Klammern dem Namen der Feldvariablen nachgestellt (auf Folie zeigen).

Die in der Feldvariablen zu speichernden Werte können dann über Einzelzuweisungen in die einzelnen, für die Feldvariable reservierten Speicherplätze geschrieben werden, z.B. also

```
frequenzen [1] = 330;
```

wenn der Wert 330 in den zweiten Speicherplatz geschrieben werden soll, der für die Feldvariable namens `frequenzen` reserviert wurde (auf Folie zeigen).

Inhaltsfolie 3 zu „Ein- und Ausgabe von Feldvariablen“:

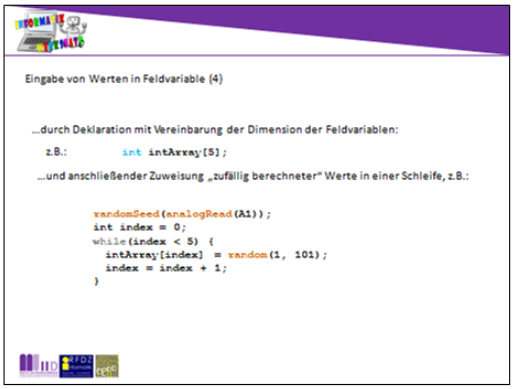


Textvorschlag:

...schneller aber geht es, Werte an Feldvariable in einer Schleife zuzuweisen.

Dies ist allerdings nur möglich, wenn die Werte „fortlaufend“ berechnet werden können...

Inhaltsfolie 4 zu „Ein- und Ausgabe von Feldvariablen“:



Textvorschlag:

...oder wenn die Werte mit dem vorprogrammierten Zufallsgenerator „zufällig berechnet“ werden.

Dabei verwenden wir folgende neue Befehle:

randomSeed(Startwert) legt einen Startwert für den Zufallsgenerator fest. Bei der Programmierung des Arduino wird als Startwert häufig der Signalwert, der an einem analogen Steckkontakt anliegt, verwendet.

random(start, ende) berechnet ausgehend vom Startwert eine „zufällige Zahl im Bereich von **start** (inklusive) bis **ende** (exklusive).

Inhaltsfolie 5 zu „Ein- und Ausgabe von Feldvariablen“:

Textvorschlag:

Die Werte, die in einer Feldvariablen gespeichert sind, können am Computerbildschirm ausgegeben werden. Die Arduino-Programmierumgebung bietet dazu als Werkzeug den seriellen Monitor an.

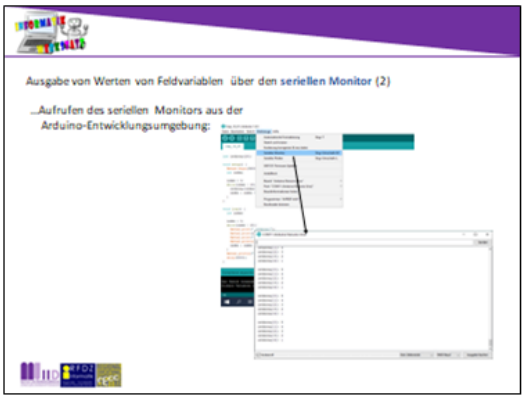
Dazu muss das Arduino-Board mit dem Computer verbunden sein und im Programm eine serielle Verbindung zwischen dem Board und dem Computer aufgebaut werden. Der Befehl dazu ist `Serial.begin(9600)`, wobei in der Klammer als Parameter die Übertragungsrate in der Einheit baud (= Anzahl der Symbole, die pro Sekunde übertragen werden) angegeben wird.

Dieser Befehl steht sinnvollerweise im Code des `setup`-Teils.

Die Ausgabe erfolgt dann mit den Befehlen `Serial.print(Text)` oder `Serial.println(Text)` wobei beim zweiten Befehl nach der Ausgabe in die nächste Zeile gewechselt wird.

Text kann eine in doppelte Hochkommata eingeschlossene Zeichenkette oder der Name einer Variablen sein.

Inhaltsfolie 6 zu „Ein- und Ausgabe von Feldvariablen“:



Textvorschlag:

In der Arduino-Programmierumgebung muss dann – wie auf der Folie gezeigt – über den Menüpunkt »Werkzeuge – Serieller Monitor« das Ein- und Ausgabefenster des seriellen Monitors geöffnet werden. Nach dem Start des Programms auf dem Arduino-Board werden dann die Werte in diesem Fenster ausgegeben.

Die angegebenen Beispiele sollen zeigen, dass Feldvariable in Kombination mit Programmschleifen mächtige Instrumente zur Datenspeicherung darstellen. Auch die „Verarbeitung“ der in Feldvariablen gespeicherten Daten (Werte) kann mit Hilfe von Programmschleifen effizient formuliert werden – Beispiele dazu finden sich in der Informationsdatei **ST_I_10Arduino_Praxis_Feldvariable** und auf der folgenden Folie:

Inhaltsfolie zu „Berechnungen mit Feldvariablen“:

Textvorschlag:

Mit Programmschleifen können Feldvariable aber nicht nur effizient mit Werten befüllt bzw. deren Werte ausgegeben werden, Programmschleifen vereinfachen auch Berechnungen mit Feldvariablen. Auf der Folie sind drei Beispiele angeführt

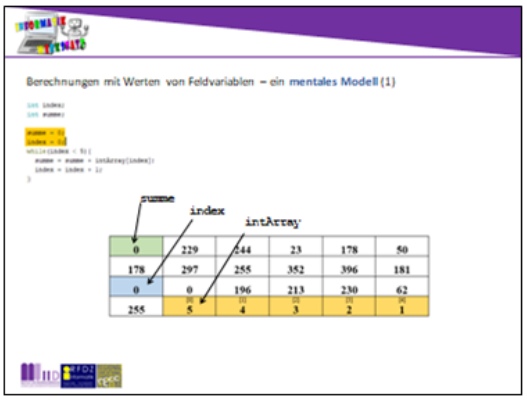
(2 x Enter-Taste zum Überblenden).

Hinweis: Die Präsentation dieser Folie ist deswegen wichtig, weil Arbeitsanregung 0 vom Aufgabenblatt darauf abzielt, diese in der Informationsdatei abgebildeten Programmfragmente jeweils zu lauffähigen Arduino-Programmen zu kombinieren. Die „Zusammenschau“ von Eingabe – Berechnung – Ausgabe auf dem seriellen Monitor kann da für die Bearbeitung der Arbeitsanregung sehr hilfreich sein.

Zudem kann beim Zeigen dieser Folie (nochmals) darauf hingewiesen werden, dass die so codierten Programme auch mit einem Arduino-Board OHNE zusätzliche Schaltung getestet werden können.

Für das eigene Codieren von Berechnungen mit Werten aus Feldvariablen ist das Verstehen des Unterschieds zwischen dem Wert des Index und dem Wert an der jeweils indizierten Speicherzelle der Feldvariablen wesentlich. Diesem Aspekt wird in der Informationsdatei und in der Kurzpräsentation durch eine detaillierte Analyse der Schritte bei der Berechnung der Summe von Feldwerten Rechnung getragen:

Inhaltsfolie 1 zur Analyse von Berechnungen mit Werten aus Feldvariablen:

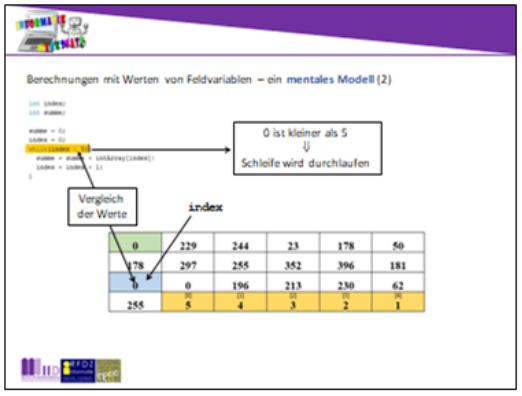


Textvorschlag:

Zum Entwickeln einer Vorstellung, wie Berechnungen mit den in einer Feldvariablen gespeicherten Werten unter Nutzung einer Programmschleife ablaufen, nutzen wir die Modellvorstellung zu Feldvariablen. Diese haben wir im Abschnitt über Arduino und Schall entwickelt und interpretieren nun beispielhaften Code anhand dieses Modells.

Bei der Deklaration der Variablen `summe`, `index` und `intArray` werden für diese passend große Speicherbereiche reserviert, in denen nach entsprechenden Wertzuweisungen an die Variablen die abgebildeten Werte „abgelegt" sind.

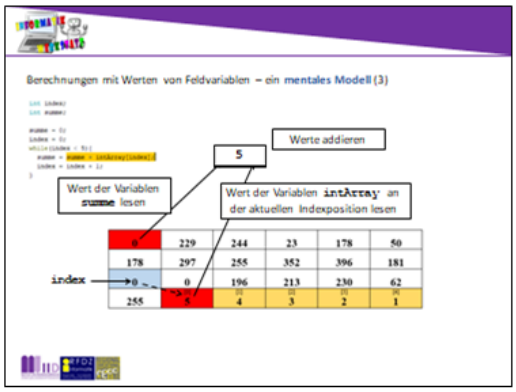
Inhaltsfolie 2 zur Analyse von Berechnungen mit Werten aus Feldvariablen:



Textvorschlag:

Zunächst wird überprüft, ob die Schleife durchlaufen werden soll. In der Schleifenbedingung wird der Wert der Variablen `index` mit der Zahl 5 verglichen. Da der in der Variablen `index` gespeicherte Wert (= 0) kleiner ist als 5, ist die Schleifenbedingung erfüllt und die Schleife wird durchlaufen.

Inhaltsfolie 3 zur Analyse von Berechnungen mit Werten aus Feldvariablen:

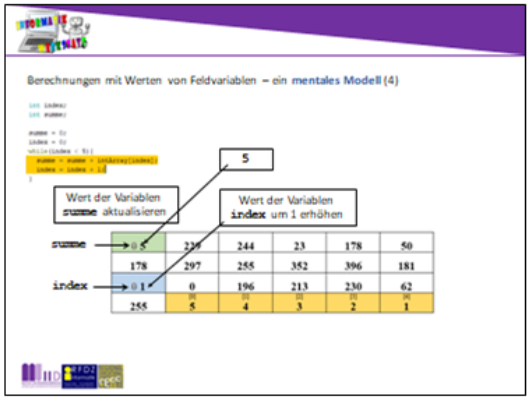


Textvorschlag:

Zuerst werden in der Programmschleife der Wert der Variablen `summe` und der Wert in jener Speicherzelle der Feldvariablen `intArray`, auf die der in der Variablen `index` gespeicherte Wert verweist, in der jeweiligen Speicherzelle gelesen (4 x Enter drücken zum Einblenden)...

...sodann werden die beiden Werte addiert (2 x Enter drücken zum Einblenden).

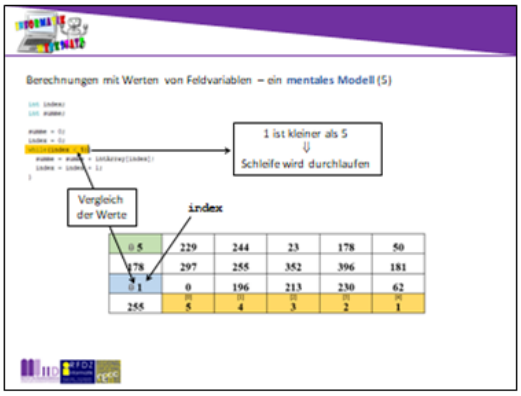
Inhaltsfolie 4 zur Analyse von Berechnungen mit Werten aus Feldvariablen:



Textvorschlag:

Durch die letzten beiden Befehle in der Programmschleife wird die errechnete Summe in die für die Variable `summe` reservierte Speicherzelle geschrieben und der Wert in jener Speicherzelle, auf die die Variable `index` verweist, um 1 erhöht.

Inhaltsfolie 5 zur Analyse von Berechnungen mit Werten aus Feldvariablen:

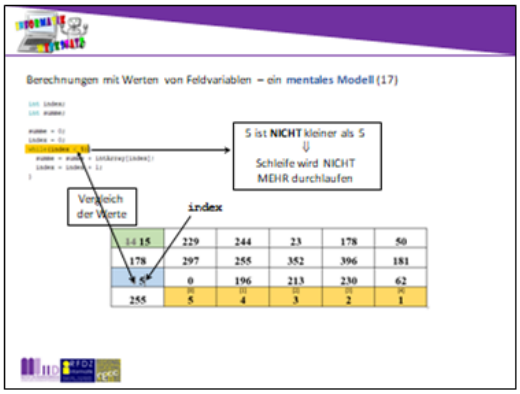


Textvorschlag:

Als nächstes wird überprüft, ob die Schleife nochmals durchlaufen werden soll. In der Schleifenbedingung wird der Wert der Variablen `index` mit der Zahl 5 verglichen. Da der in der Variablen `index` gespeicherte Wert (= 1) kleiner ist als 5, ist die Schleifenbedingung erfüllt und die Schleife wird durchlaufen.

…plus weitere 13 Folien für die restlichen Durchläufe der Programmschleife mit dem mentalen Modell der „rechteckigen Speicherzellen“, bis zu:

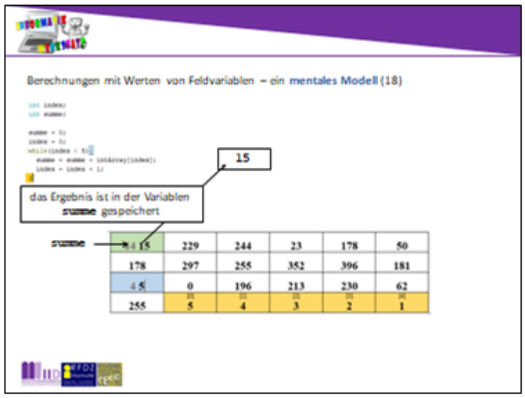
Inhaltsfolie 17 zur Analyse von Berechnungen mit Werten aus Feldvariablen:



Textvorschlag:

Als nächstes wird überprüft, ob die Schleife nochmals durchlaufen werden soll. In der Schleifenbedingung wird der Wert der Variablen `index` mit der Zahl 5 verglichen. Da der in der Variablen `index` gespeicherte Wert (= 5) NICHT kleiner ist als 5, ist die Schleifenbedingung NICHT erfüllt und die Schleife wird NICHT MEHR durchlaufen.

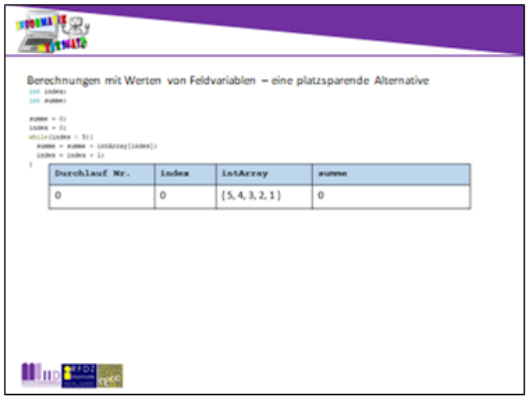
Inhaltsfolie 18 zur Analyse von Berechnungen mit Werten aus Feldvariablen:



Textvorschlag:

Das Ergebnis der Berechnung ist nun der in der Variablen `summe` gespeicherte Wert…

…und Inhaltsfolie 19 zur Analyse von Berechnungen mit Werten aus Feldvariablen mit platzsparender Tabellierung der aktuellen Variablenwerte.



Textvorschlag:

Eine platzsparende Alternative zu der in den vorangegangenen Folien vorgestellten Versinnbildlichung bei Berechnungen mit Werten von Feldvariablen bieten Tabellen.

Dabei werden für jeden Schleifendurchlauf die Werte aller relevanten Variablen notiert bzw. aktualisiert…

(sechs mal Enter drücken um die Tabelle Zeile für Zeile sichtbar zu machen…)

Hinweis: Auch wenn das Durchbesprechen des gesamten Beispiels bzw. das Durcharbeiten der gesamten Arbeitsanregungen zur Analyse und Veranschaulichung von Berechnungen mit Werten von Feldvariablen etwas zeitintensiver ist, sollte die Zeit für das vollständige Durchdenken aller Berechnungsschritte bereitgestellt werden. Dadurch kann die notwendige Sicherheit im Umgang mit Feldvariablen erlangt werden, die für das **Codieren eigener Befehle und eigener Fragen** mit Feldvariablen als Parameter notwendig ist.

Dabei ist das Codieren **eigener Befehle** aus den Dateien von Abschnitt 6 im zweiten Arbeitspaket schon bekannt:

Der Code eines **Programmblocks**, der einen eigenen **Befehl** definiert, hat folgendes Aussehen:

```
void befehlName(Datentyp parameter1, Datentyp parameter2,…) {
  programmBefehl1;
  programmBefehl2;
  ...
}
```

Die Parameter – entsprechend den Daten, die der Befehl »von außen« benötigt – können auch als Feldvariable übergeben werden. Der durch nebenstehenden Code definierte Befehl gibt beispielsweise die Werte einer Feldvariablen, die als Parameter namens **intArray** übergeben wird, am seriellen Monitor aus:

```
void befehlPrintIntArray(int intArray[], int dim){
  int index;

  index = 0;
  while(index < dim){
    Serial.print("Wert an Indexposition ");
    Serial.print(index);
    Serial.print(": ");
    Serial.println(intArray[index]);
    index = index + 1;
  }
}
```

Wenn der selbst codierte Programmblock aber nicht nur eine Aktion ausführen soll, sondern – z.B. als Ergebnis einer Berechnung – eine „Antwort“ liefern soll, ist es anschaulich naheliegender, von einer **eigenen Frage** zu sprechen.

Der Code eines **Programmblocks**, der eine eigene **Frage** definiert, hat folgendes Aussehen:

```
int frageName(Datentyp parameter1, Datentyp parameter2,…) { // !!
  int antwort;                                              // !!
  programmBefehl1;
  programmBefehl2;
  ...
  return antwort;                                           // !!
}
```

Die auffälligsten **Unterschiede zu** einem **Befehls-Programmblock** sind:

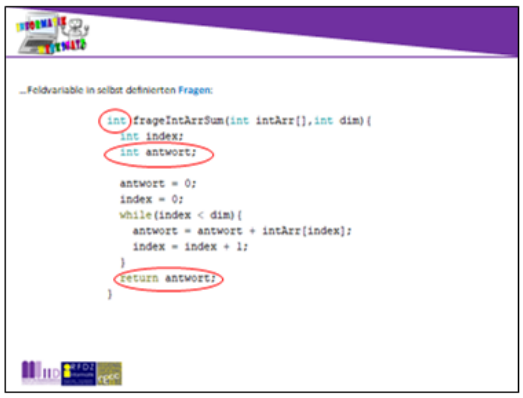
Der Datentyp des Programmblocks ist nicht `void` sondern muss gleich jenem Datentyp sein, von dem auch die Antwort ist! Im Beispiel zeigt der Datentyp `int` des Programmblocks an, dass eine ganzzahlige Antwort gegeben werden muss!

Sinnvollerweise wird innerhalb des Programmblocks eine Variable deklariert, die die Antwort „zwischenspeichert". Diese Antwortvariable heißt in obigem Beispiel **`antwort`** und muss verständlicherweise vom Datentyp `int` sein…

Mit dem Befehl **`return`** schließlich wird der zuvor in der Antwortvariablen gespeicherte Wert „als Antwort“ „zurückgegeben“.

…und: Mit der Antwort eines Frage-Programmblocks muss „etwas passieren“, d.h. die Antwort bzw. der Wert, der als Antwort gegeben wurde, muss in einer weiteren Variablen passenden Datentyps gemerkt werden oder er muss ausgegeben werden oder…:

Inhaltsfolie 1 zu Frage-Programmblöcken mit Feldvariablen:



Textvorschlag:

Alternativ zu Befehlen können auch **Fragen** als „Unterprogramme“ selbst definiert werden, insbesondere wenn ein berechneter Wert als „Antwort gegeben“ werden soll.

Dazu muss in der Signatur anstelle von `void` der Typ der jeweiligen Antwort (hier: `int`, weil die Antwort eine ganze Zahl ist) geschrieben werden
(auf Folie zeigen)

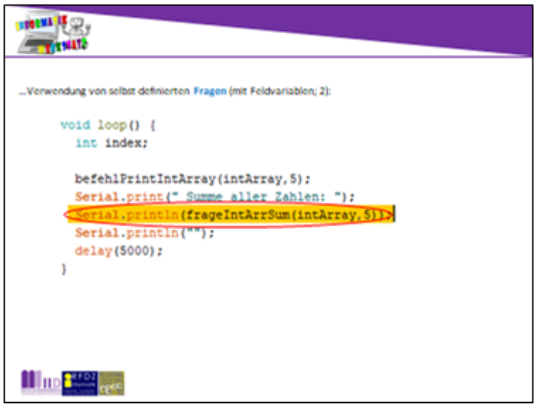
Zudem muss im Programmcode für die Frage eine Variable dieses Typs (hier: **`antwort`**) deklariert werden, deren Wert als letzter Befehl im Programmcode mit dem Schlüsselwort **`return`** (eben als „Antwort“) „zurückgegeben“ wird).

Inhaltsfolie 2 zu Frage-Programmblöcken mit Feldvariablen:

Textvorschlag:

Wenn eine solche selbst-definierte Frage dann (z.B. im `loop`-Programmteil) verwendet wird, muss die „Antwort" von einer Variablen des selben Datentyps (hier: `summe`) „entgegen genommen" genommen werden (auf Folie auf die gekennzeichneten Codezeilen zeigen)...

Inhaltsfolie 3 zu Frage-Programmblöcken mit Feldvariablen:



Textvorschlag:

...alternativ kann aber die „Antwort" des „Frage-Unterprogramms" auch direkt auf dem seriellen Monitor ausgegeben werden (auf markierte Codezeile zeigen).

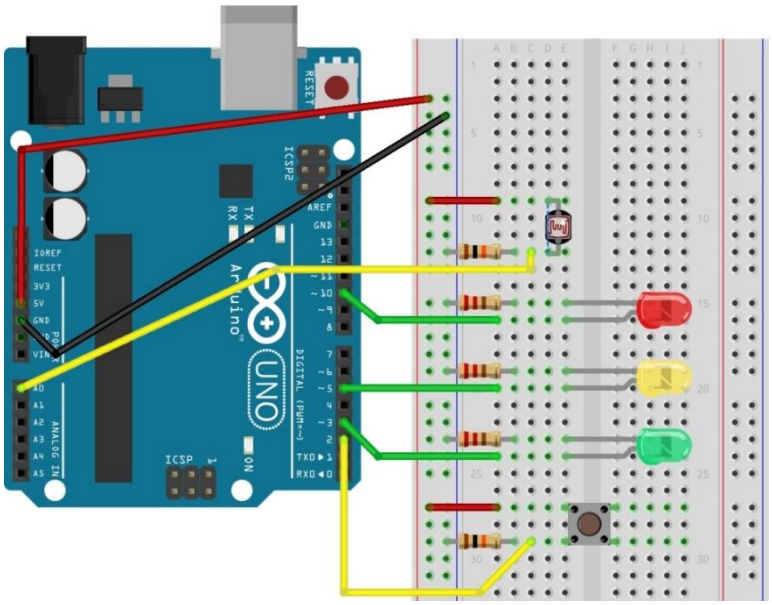
Die vier Arbeitsanregungen zu diesem Abschnitt weisen eine Besonderheit auf, indem sie als **Aufgabe 0)** bis **Aufgabe 3)** nummeriert sind. Der Hintergrund hinter dieser ungewohnten Nummerierung ist, dass die erste Arbeitsanregung lediglich die Neukombination bzw. die individuelle Veränderung von den in der Informationsdatei abgebildeten Codefragmenten beinhaltet. Da hierbei individuelle Lösungen erwartet werden dürfen, ja sogar erwünscht sind, existiert zu dieser **Aufgabe 0)** auch kein Lösungsvorschlag in der Lösungsdatei **ST_LO_10Arduino_Praxis_Feldvariable**.

In den nächsten beiden Arbeitsanregungen wird das Analysieren von Berechnungen mit Werten aus Feldvariablen eingefordert. Unter anderem ist auch Code zur Berechnung des arithmetischen Mittels der in einer Feldvariablen gespeicherten Werte zu analysieren. Dabei wird innerhalb der Aufgabenstellung nochmals auf die Besonderheit der Ganzzahldivision in Sprachen der C-Familie eingegangen, auf die auch in der Informationsdatei in einer Bemerkung hingewiesen wird.

In der letzten Aufgabe sind (teilweise bereits bekannte) Berechnungen mit Werten von Feldvariablen als Befehls- oder Frage-Programmblöcke zu codieren und zu testen. Diese Übung wird im nächsten Abschnitt fortgesetzt und dort mit der Codierung von Programmbibliotheken abgerundet.

Arduino – Programmierpraxis: Messwerterfassung

Feldvariable sind gut geeignet, Messwerte von Sensoren zu speichern. Ausgangspunkt ist dabei die aus Abschnitt 8 bekannte Ampelschaltung mit Photowiderstand (vgl. Abb. bzw. auch die erweiterte Variante mit „Fußgängerampel"):

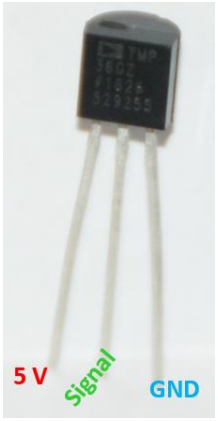


Anhand eines Programmfragments wird in der Informationsdatei **ST_I_11Arduino_ Praxis_Messwerterfassung** prinzipiell gezeigt, wie die Messwerte in eine Feldvariable gespeichert werden können. Bewusst ist dabei der Code so gestaltet, dass bei jedem Durchlauf des `loop`-Programmblocks nur ein Messwert verarbeitet wird. Dadurch kann durch Einbau eines zusätzlichen `delay`-Befehls dafür gesorgt werden, dass die Messungen zeitlich nicht zu nahe beieinander liegen, ohne dass andere Befehle im `loop`-Block blockiert werden, wie dies bei Codierung mit einer Programmschleife auftreten kann:

```
int intArray[5];
int index =0;


void loop() {
  // Befüllen eines int-Feldes mit Messwerten
  // index wird außerhalb deklariert
  // und mit 0 initialisiert

  sensorValue = analogRead(A0);
  if(index < 5){
    intArray[index] = sensorValue;
    index = index + 1;
  }
  else{
    // das Feld ist mit Werten befüllt
    // und kann im else-Zweig über den
    // seriellen Monitor ausgegeben werden
  }
```

Zudem kann dieser Code einfach für Messwerterfassung mit anderen Sensoren angepasst werden. In diesem Abschnitt wird dies beispielhaft gezeigt für den

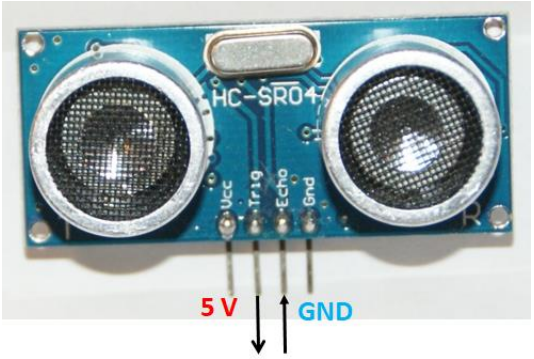
- den TMP36-Temperatursensor, dessen Sensorleitung wie jene des Photowiderstandes an einen analogen Port (z.B. A1) angeschlossen werden muss:



Bei der Messwerterfassung sind zusätzlich lediglich die Spannungswerte in Temperaturwerte umzurechnen, z.B. so:

```
//collecting data
if(index < 100){
  //lighting the green LED
  //to indicate cata collection
  digitalWrite(greenLedPin,HIGH);
  sensorValue = analogRead(A1);
  voltage = (sensorValue/1024.0)*5.0;
  tempC = (voltage - 0.5008)*100;
```

- den HC-SR04-Ultraschallsensor:



Die beiden hier für die Messwerterfassung verwendeten, mit „Trig“ und „Echo“ beschrifteten Kontakte sind mit digitalen Ports zu verbinden, die im Modus `Output` (für den „Trig“-Kontakt) bzw. im Modus `Input` (für den „Echo“-Kontakt) betrieben werden müssen. Bei der Abstandsmessung wird über den „Trig“-Kontakt eine Ultraschallwelle ausgesandt und nach deren Reflexion an einem Objekt wieder detektiert. Über den „Echo“-Kontakt wird dann die Laufzeit der Welle (in Mikrosekunden) gemessen. Die Codierung dieses Messverfahrens kann beispielsweise so erfolgen:

```
//sensor management
int triggerPin = 8;
int echoPin =9;

  void setup() {
    Serial.begin(9600);
    pinMode(triggerPin, OUTPUT);
    pinMode(echoPin, INPUT);
```

```
void loop() {
  long timePassed;
  float distance;
  long sum;
  float mean;
  int minVal;
  int maxVal;

 //collecting data
   if(index < 100){
   //lighting the green LED
   //to indicate cata collection
   digitalWrite(greenLedPin,HIGH);
   digitalWrite(triggerPin,LOW);
   delay(5);
   digitalWrite(triggerPin,HIGH);
   delay(10);
   digitalWrite(triggerPin,LOW);
   timePassed = pulseIn(echoPin,HIGH);
   distance = timePassed*0.03432/2;
```

Tatsächlich stellt die Messwerterfassung über Sensoren aber nur eine weitere Möglichkeit der Dateneingabe in Feldvariable dar. Übungsschwerpunkt stellt auch in diesem Abschnitt das aus dem vorangegangenen Abschnitt bekannte Codieren von Befehl- und Frage-Programmblöcken (zur Berechnung statistischer Maßzahlen) mit Feldvariablen als Parametern dar.

Die Beobachtung, dass beim Lösen der gestellten Aufgaben diese Programmblöcke immer gleichbleiben, in jedem neuen Programmierprojekt aber codiert werden müssen, motiviert das Erstellen einer „Sammlung von eigenen Befehlen und Fragen“, die in das jeweilige Programmierprojekt eingebunden werden kann, wo die Programmblöcke aber eben nur einmal codiert werden müssen.

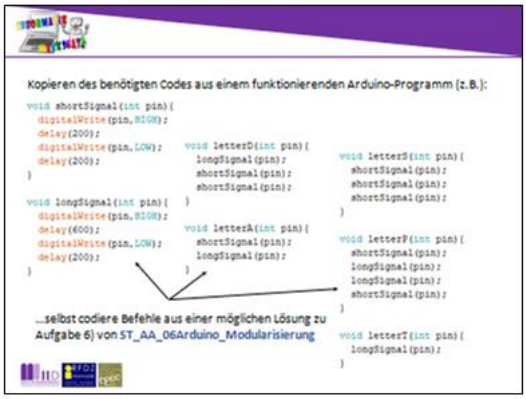
Eine solche „Sammlung von eigenen Befehlen und Fragen“ nennt man **Programmbibliothek**. Eine Programmbibliothek besteht in der Programmiersprache C aus **zwei Dateien**:

Eine Datei – mit der Dateiendung **.c** oder **.cpp** – beinhaltet den **Code aller Befehls- und Frage-Blöcke**, die durch die Programmbibliothek angeboten werden, und den Code jener Programmblöcke, die von den bereitgestellten Befehls- und Frage-Blöcke als „Hilfsprogramme“ benötigt werden.

Eine sogenannte **„Header-Datei“** – mit der Dateiendung **.h** – beinhaltet nur die „Überschriften“, man sagt auch: die **Signaturen**, der Befehls- und Frage-Blöcke, die „öffentlich“ zur Verfügung gestellt werden. Die Signatur eines Programmblocks beinhaltet also den Datentyp des Programmblocks, den Namen des Programmblocks und die Parameterliste des Programmblocks.

Obwohl das Codieren der Programmbibliothek sich im Wesentlichen auf das Umkopieren von bereits vorhandenem Code beschränkt, ist dabei einiges zu beachten, nicht zuletzt wegen des notwendigen Wechsels von der gewohnten Arduino-Programmierumgebung zu einem Texteditor, also einem anderen Anwendungsprogramm. Daher wird dies sowohl in der Informationsdatei **ST_I_ 11Arduino_Praxis_Messwerterfassung** als auch in der Kurzpräsentation **ST_PR_ 11Arduino_Programmbibliotheken** – zunächst am Beispiel der Programmblöcke für den „Morseapparat“ ausführlich erklärt:

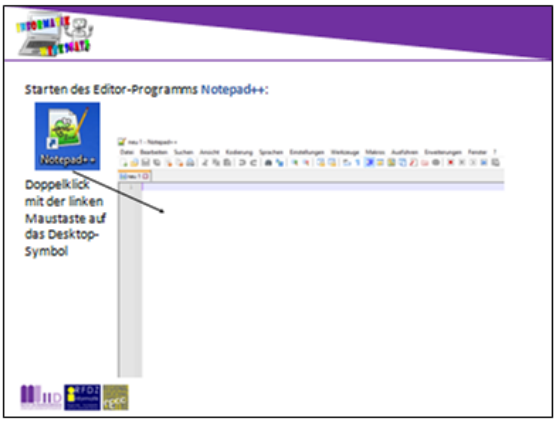
Inhaltsfolie 1 zur Codierung einer Programmbibliothek:



Textvorschlag:

Beim Codieren von Arduino-Programmbibliotheken ist es empfehlenswert, die Befehle und/oder Fragen, die die Programmbibliothek enthalten soll, zuerst wie gewohnt in einem Arduino-Programmierprojekt zu codieren und zu testen, und erst danach den benötigten Code in eine Programmbibliothek (mit der Dateiendung .cpp) zu kopieren.

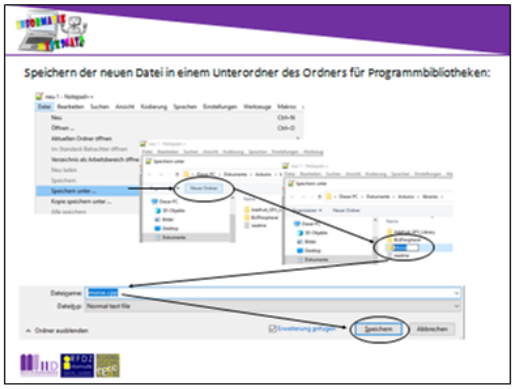
Inhaltsfolie 2 zur Codierung einer Programmbibliothek:



Textvorschlag:

Da in der gewohnten Arduino-Entwicklungsumgebung die Programmbibliotheks-Dateien nicht editiert werden können, verwenden wir dazu das Editorprogramm Notepad++.

Inhaltsfolie 3 zur Codierung einer Programmbibliothek:



Textvorschlag zur Erklärung (die Einblendungen erfolgen jeweils nach dem Drücken der <Enter>-Taste:

Nach Auswahl des Menüpunktes „Date - Speichern unter…" → Einblendung …

…durch Klicken auf die Schaltfläche „neuer Ordner" einen Ordner für die Programmbibliothek erstellen → Einblendung …

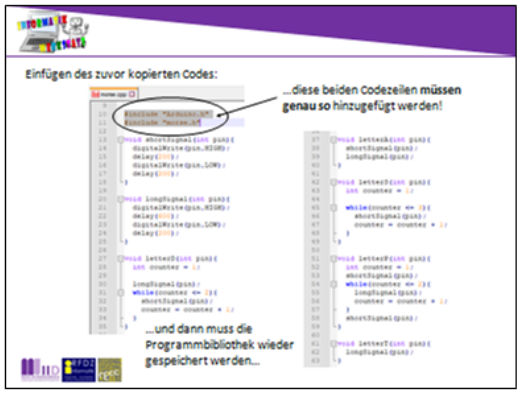
…und diesem neuen Ordner einen aussagekräftigen (gültigen) Namen geben. → Einblendung …

Sodann einen aussagekräftigen Namen **mit Dateiendung .cpp** für die Bibliotheksdatei vergeben → Einblendung …

…und die Datei dann durch Klicken auf die Schaltfläche „Speichern" speichern.

Anmerkung: Es sollte darauf hingewiesen werden, dass alle Programmbibliotheken in einem Unterordner von Dokumente/Arduino/libraries des jeweils angemeldeten Benutzers gespeichert werden, da sie ansonsten „nicht gefunden" werden!

Inhaltsfolie 4 zur Codierung einer Programmbibliothek:



Textvorschlag:

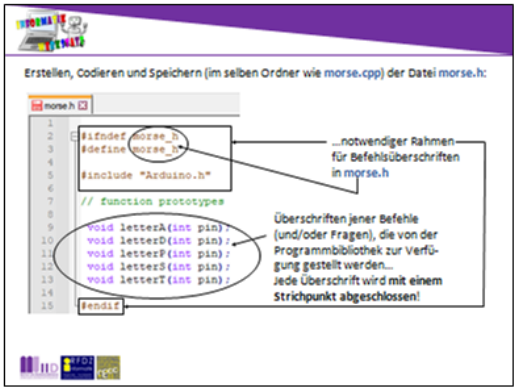
Der eingefügte Code muss noch um zwei Codezeilen ergänzt werden:

In jeder Arduino-Programmbibliothek ist die Anweisung **#include "Arduino.h"** hinzuzufügen.

Wenn die Programmbibliothek xxxx.cpp genannt wurde, ist zudem noch die Anweisung **#include "xxxx.h"** hinzuzufügen.
(da die gezeigte Programmbibliothek **morse.cpp** genannt wurde, ist hier also **#include "morse.h"** hinzugefügt).

Zur Erläuterung:
Die Dateiendung **.h** zeigt an, dass es sich um eine sogenannte **Header-Datei** handelt. In dieser Header-Datei sind die Namen der Befehle bzw. Fragen aufgelistet, die in der Programmbibliothek codiert sind UND die außerhalb der Programmbibliothek verwendet werden können.
Während die Datei **Arduino.h** standardmäßig in der Arduino-Umgebung vorhanden ist, muss die header-Datei **morse.h** für die Programmbibliothek **morse.cpp** erst codiert werden. Dies ist Inhalt der nächsten Folie.

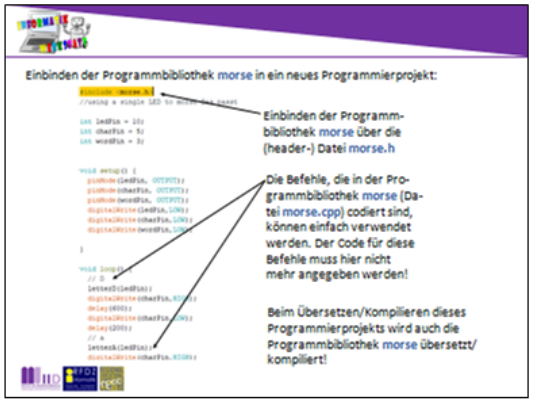
Inhaltsfolie 5 zur Codierung einer Programmbibliothek:



Textvorschlag:

Über den Menüpunkt „Datei – neu“ ist nun noch eine Datei zu erstellen, die im selben Ordner wie die Datei **morse.cpp** unter dem Namen **morse.h** gespeichert werden muss.
In dieser Datei sind die Überschriften (man sagt auch: Signaturen) aller Befehle und/oder Fragen anzugeben, die in der Programmbibliothek (**morse.cpp**) codiert sind und nach dem Einbinden der Programmbibliothek in einem anderen Arduino-Projekt zur Verfügung stehen sollen. Zum Beispiel sind in der Datei morse.h die Überschriften der Befehle shortSignal bzw. longSignal nicht angegeben – diese Befehle können daher nur innerhalb der Programmbibliothek, nicht aber „von außen“ verwendet werden.

Inhaltsfolie 6 zur Codierung einer Programmbibliothek:



Textvorschlag

Wenn die in einer Programmbibliothek codierten Befehle und/oder Fragen in einem neuen Programmierprojekt verwendet werden sollen, ist lediglich am Anfang des Programms (in der Arduino-Programmierumgebung) die jeweilige Programmbibliothek über die entsprechende Header-Datei einzubinden.

Beim Übersetzen/Kompilieren dieses Programmierprojekts wird auch die Programmbibliothek kompiliert – beim ersten Kompilieren kann es da zu Fehlermeldungen kommen, die den Code der Programmbibliothek betreffen. Die Vorgangsweise in diesem Fall wird auf den nächsten Folien gezeigt.

Inhaltsfolie 7 zur Codierung einer Programmbibliothek:

Textvorschlag:

...siehe Text auf Folie

Bemerkung: Eine Änderung der Arduino-Voreinstellungen, sodass detaillierte Informationen während des Kompilierens angezeigt werden, wird nicht empfohlen, da nach dem Einfügen der detaillierteren Fehlermeldungen in eine neue Datei des Notepad++-Editors praktischerweise sowohl die .cpp-Quelldatei und die Datei mit der Fehlermeldung im selben Editor geöffnet werden können.

Inhaltsfolie 8 zur Codierung einer Programmbibliothek:

Textvorschlag:

Die detaillierte Fehlermeldung beinhaltet eine Angabe, um welchen Fehler es sich handelt und WO ein Fehler BEMERKT wurde, d.h. die angegebene Position – hier 40:1, d.h. in Zeile 40, erstes Zeichen – gibt diejenige Position an, unmittelbar vor der der Fehler aufgetreten ist (im abgebildeten Beispiel wurde der Strichpunkt am Ende der Zeile 39 vergessen...)

Praxis Elektrizität: Spannung und Widerstand

Als Erweiterung des Grundwissens zur Elektrizität wird aufbauend auf die Vorstellung von „im-Kreis-strömender-Elektrizität, die Energie transportiert“ (vgl. Arbeitspaket 1) eine mit **elektrischem Druck-unterschied** eine tragfähige **Analogie für die elektrische Spannung** entwickelt. Diese Analogie ist physikalisch vertretbar,

- treiben doch (mechanische) Druckunterschiede als Energie pro Volumen Volumenströmungen an,
- während elektrische Spannungen, d.h. elektrischer Druckunterschiede, als Energie pro Ladung Strömungen von Ladungen antreiben.

Bewusst wird aber der Begriff der elektrischen Ladung vermieden und als „Elektrizität“ umschrieben, da damit alle Probleme, die die Unterscheidung von negativer und positiver Ladung mit sich bringt (z.B. die Frage nach der Richtung des elektrischen Stroms), ausgeblendet werden können, ohne dass das Verständnis für die grundlegenden Gegebenheiten und Eigenschaften „der Elektrizität“ dadurch behindert wird.

Durch die Analogie ist leicht verständlich, dass

- die elektrische Spannung (als Druckunterschied) stets zwischen zwei Punkten zu messen ist,
- gemäß den Gesetzmäßigkeiten kommunizierender Gefäße die elektrische Spannung (als Druck-unterschied) in parallelen Zweigen einer Schaltung gleich groß ist.

Zudem gestattet diese Analogie die unkomplizierte Festlegung, dass (elektrischer, d.h. Ohm‘scher) **Widerstand** etwas ist, das bei gleichem (elektrischem) Druckunterschied unterschiedlich starken (elektrischen) Strom bewirkt, was

- das **Ohm’sche Gesetz** gemäß: $\text{elektrische Stromstärke} = \frac{\text{elektrische Spannung}}{\text{(Ohm'scher) Widerstand}}$
- und ebenso die **Spannungsteilerregel**, dass bei konstanter elektrischer Stromstärke durch hinter-einandergeschaltete Widerstände am höheren Widerstand eine größere elektrische Spannung anliegen muss,

unmittelbar einsichtig macht. Für detailliertere Ausführungen sei auf die Informationsdatei **ST_I_12 Elektrizität_Praxis_Spannung_Widerstand** bzw. auf die entsprechenden Folien der Kurzpräsentation **ST_PR_12Elektrizität_Spannung_Widerstand** verwiesen:

Inhaltsfolie 1 zu „Spannung und Widerstand“:

Textvorschlag:

Was treibt die Elektrizität an?

Wir verwenden dazu eine weitere Analogie und betrachten anstelle des elektrischen Stroms einen Luft-Strom:

Beim Aufpumpen eines Fahrradreifens wird Luft in das Innere des Reifens gedrückt.

(Enter-Taste zum Einblenden)

Nach dem Aufpumpen herrscht im Inneren des Reifens in alle Richtungen der gleiche Luftdruck. Dieser Druck im inneren des Reifens ist höher als der Luftdruck außerhalb des Reifens.

(Enter-Taste zum Einblenden)

Durch das Aufpumpen entsteht ein Druckunterschied, der nach dem Öffnen des Ventils einen Luft-Strom antreibt.

(2 x Enter-Taste zum Einblenden)

Inhaltsfolie 2 zu „Spannung und Widerstand“:

Textvorschlag:

Sicher ist dir klar, was passiert...

(Enter-Taste zum Einblenden)

...wenn ein (z.B.) ein Wattebausch vor das Ventil des Fahrradschlauches gedrückt wird?

(evtl. Reaktionen/Antworten abwarten, dann Enter-Taste zum Einblenden)

(zur Bestätigung bzw. Korrektur der Reaktionen/Antworten)

(falls vorhanden kann dieses „Analogieexperiment“ auch mit Pumpe, Schlauch und Wattebausch vor Ort durchgeführt werden)

Inhaltsfolie 3 zu „Spannung und Widerstand“:

Textvorschlag:

Wir dürfen uns in Analogie zum Luftstrom aus dem Fahrradreifen vorstellen, dass elektrischer Strom durch einen elektrischen Druckunterschied hervorgerufen wird.

Dieser Druckunterschied wird durch eine Spannungsquelle erzeugt: An dem Ende einer Batterie, das durch ein „+“ gekennzeichnet ist, stellt man sich elektrischen Überdruck vor, am anderen Ende, das durch ein „–“ gekennzeichnet ist, elektrischen Unterdruck.

So wie im Reifen überall derselbe Luftdruck herrscht, dürfen wir uns vorstellen, dass in allen Punkten eines Stromkreises, die mit einem der beiden Pole der Batterie verbunden sind, der gleiche elektrische Druck herrscht.

(2 x Enter-Taste zum Einblenden).

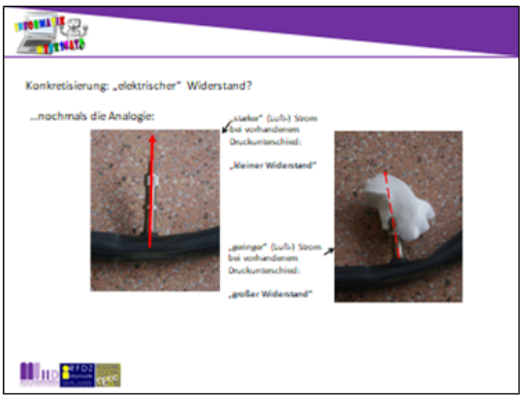
In den Abbildungen sind diese Druckverhältnisse durch die Farben, die wir von den Steckplatinen kennen, versinnbildlicht: Je höher der elektrische Druck ist, desto intensiver sind die Farben gewählt.

(Enter-Taste zum Einblenden)

Elektrische Spannung bedeutet nach dieser Vorstellung nichts anderes als elektrischen Druckunterschied – und wird stets zwischen zwei Punkten gemessen.

(4 x Enter-Taste zum zweimaligen Ein- und Ausblenden)

Inhaltsfolie 4 zu „Spannung und Widerstand“:



Textvorschlag:

Im Verlauf der Arbeiten mit den Arduino-Schaltungen ist immer wieder der Begriff „Widerstand“ bzw. „Ohm‘scher“ Widerstand verwendet worden. Zur Entwicklung einer tragfähigen Vorstellung, was im Zusammenhang mit Elektrizität unter „Widerstand“ zu verstehen ist, bemühen wir nochmals die Analogie zwischen elektrischem Druckunterschied und dem Druckunterschied an einem Fahrradschlauch, der einen Luftstrom verursacht:

(Enter-Taste zum Einblenden)

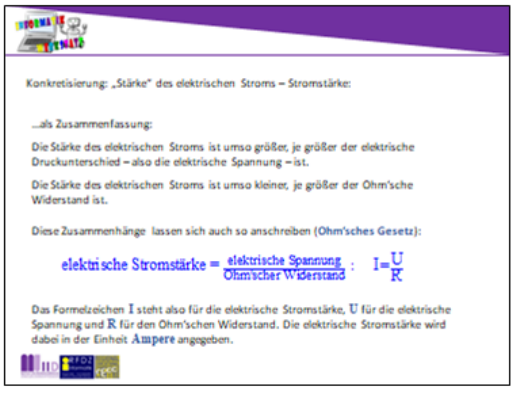
Wenn bei einem bestimmten Druckunterschied ein „starker“ Luftstrom herrscht, sprechen wir von einem „kleinen Widerstand“, den die Strömung überwinden muss.

(Enter-Taste zum Einblenden).

Wenn aber bei gleichem Druckunterschied nur ein „geringer“ Luftstrom zustande kommt, sagen wir, dass die Strömung einen „großen Widerstand“ überwinden muss.

Genauso verstehen wir auch bei der Elektrizität unter „Widerstand“ etwas, das den Elektrizitätsstrom mehr oder weniger behindert, d.h. dazu führt, dass der elektrische Strom stärker oder weniger stark ist.

Inhaltsfolie 5 zu „Spannung und Widerstand“:



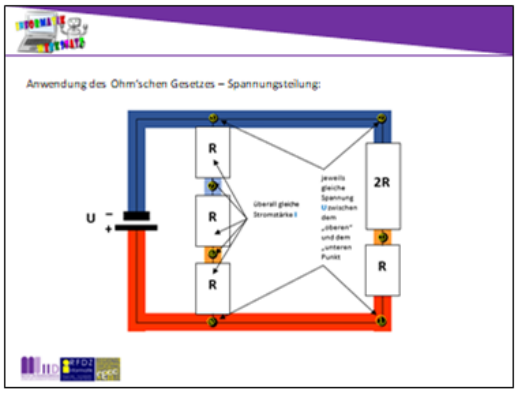
Textvorschlag:

Mit Hilfe der Vorstellungen von elektrischer Spannung und dem (Ohm‘schen) Widerstand können wir schließlich auch die Stärke des elektrischen Stroms festlegen:

(Vorlesen der beiden Absätze „Die Stärke des elektrischen Stroms....“ sowie der Darstellung des Ohm‘schen Gesetzes)

(zur Erläuterung des Ohm‘schen Gesetzes kann das Einsetzen ausgewählter Zahlenwerte hilfreich sein, z.B.: 60 Volt für die Spannung und dann der Reihe nach 5, 10, 15, 20, 30 Ohm für den Widerstand etc.; allenfalls kann – je nach Vorwissen der Zielgruppe – auch eine Umformulierung des Ohm‘schen Gesetzes auf U = ... oder R = ... erfolgen)

Inhaltsfolie 6 zu „Spannung und Widerstand“:



Textvorschlag:

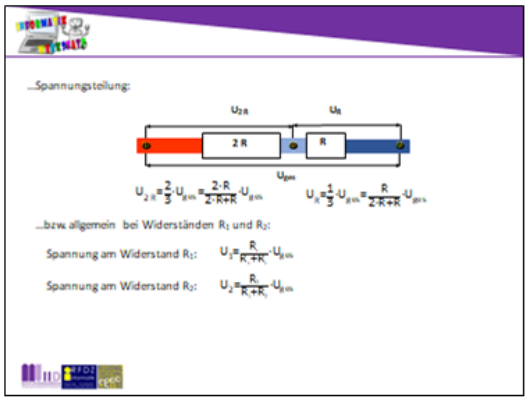
Mit dem Ohm‘schen Gesetz verstehen wir, dass wir durch Hintereinanderschaltung von Widerständen die vorhandene Spannung aufteilen können:

Wir wissen, dass der elektrische Druckunterschied, die Spannung, zwischen den mit ① und ② bezeichneten Punkten genau so groß ist wie die Spannung zwischen den mit ④ und ⑤ bezeichneten Punkten, und zwar so groß wie die Spannung, die von der eingezeichneten Spannungsquelle zur Verfügung gestellt wird. Nehmen wir an, diese Spannung beträgt fünf Volt (5 V).

Wir können uns auch leicht überlegen, dass die Stärke des elektrischen Stroms, der vom mit ① bezeichneten Punkt zu dem mit ② bezeichneten Punkt fließt, in jedem der dazwischenliegenden Punkte gleich groß sein muss (evtl. auf Folie zeigen).

Allerdings ist der Ohm‘sche Widerstand zwischen den mit ③ und② bezeichneten Punkten doppelt so groß wie der, der zwischen den mit ① und ③ bezeichneten Punkten liegt (evtl. auf Folie zeigen). Nach dem Ohm‘schen Gesetz muss daher auch der elektrische Druckunterschied, die Spannung, zwischen den mit ③ und② bezeich-neten Punkten doppelt so groß sein wie zwischen den mit ① und ③ bezeichneten Punkten...

Inhaltsfolie 7 zu „Spannung und Widerstand“:



Textvorschlag:

Wenn zwei Widerstände in Serie geschaltet sind, von denen der eine einen doppelt so hohen Widerstandswert hat wie der andere, müssen am „doppelt so großen“ Widerstand zwei Drittel der Gesamtspannung anliegen, am anderen Widerstand muss nur ein Drittel der Gesamtspannung anliegen.

(2 x Enter-Taste zum Einblenden)

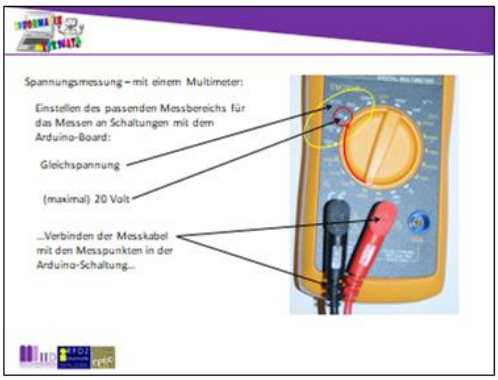
Man erhält den Wert der an einem Widerstand anliegenden Spannung, indem man das Verhältnis des Widerstandswertes des betreffenden Widerstandes durch die Summe der Widerstandswerte aller in Serie geschalteten Widerstände dividiert und diese Zahl dann mit der gesamten zur Verfügung stehenden Spannung multipliziert.

(1 x Enter-Taste zum Einblenden)

Dies gilt auch ganz allgemein bei beliebigen Verhältnissen der Widerstandswerte (allenfalls Formeln verlesen, mit Zahlen durchspielen)

Sowohl (elektrische) Spannung als auch (Ohm'scher) Widerstand werden dabei sogleich als messbare Größen eingeführt, wobei für beide Größen jeweils zwei unterschiedliche „Messverfahren" angegeben werden:

Inhaltsfolie 1 zur Spannungsmessung:



Textvorschlag:

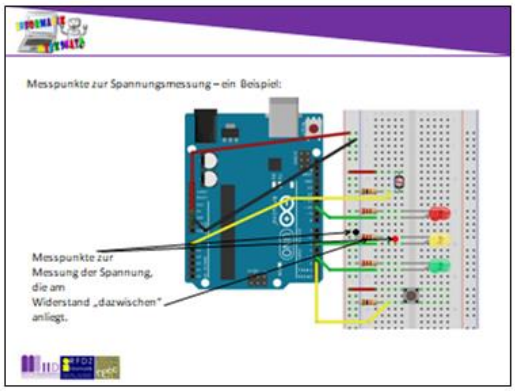
Spannung kann zum Beispiel mit einem Multimeter gemessen werden, das ist ein Messgerät, mit dem Wahlweise die elektrische Spannung, die elektrische Stromstärke oder der Ohm'sche Widerstand eines Bauteils gemessen werden kann.

Zur Spannungsmessung an einer Schaltung, die durch ein Arduino-Board mit Spannung versorgt wird, ist zunächst der Messbereich auf 20 Volt Gleichspannung

(2 x Enter-Taste zum Einblenden)

einzustellen, sodann sind das rote und das schwarze Messkabel mit den Messpunkten auf der Schaltung zu verbinden (Wechsel zu nächster Folie)

Inhaltsfolie 2 zur Spannungsmessung:



Textvorschlag:

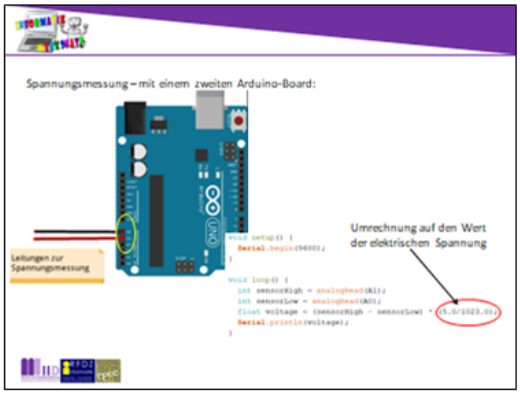
(Enter-Taste zum Einblenden)

Die mit dem roten bzw. dem schwarzen „Punkt" markierten Steckkontakte sind die Messpunkte, in die die Mess-Enden der Kabel vom Multimeter gesteckt werden. Im gezeigten Beispiel, wird der elektrische Druckunterschied zwischen einem Punkt „vor" und einem Punkt „nach" dem zwischen den Messpunkten liegenden (Ohm'schen) Widerstand gemessen.

(Enter-Taste zum Einblenden).

Man nennt diese Spannung die Spannung, die an dem Widerstand anliegt...
...oder auch die Spannung, die an dem Widerstand abfällt.

Inhaltsfolie 3 zur Spannungsmessung:



Textvorschlag:

Wenn kein Multimeter zur Verfügung steht, kann auch mit Hilfe eines zweiten Arduino-Boards leicht ein Spannungsmessgerät „gebaut" und programmiert werden:

Die Messkabel werden an zwei analoge Steckkontakte (in der Abbildung A0 und A1) angeschlossen und können so Werte des „elektrischen Drucks" messen, die – wie gewohnt bei analogen Pins – als ganze Zahlen von 0 bis 1023 codiert sind.

(Enter-Taste zum Einblenden)

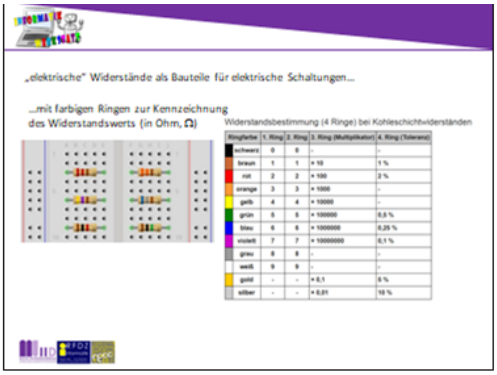
Die Differenz dieser codierten Messwerte wird dann in die Spannung umgerechnet, die zwischen den Messpunkten anliegt, und wird im Beispielprogramm über den seriellen Monitor auf einem Computer-Bildschirm ausgegeben.

...es empfiehlt sich, dieses selbst programmierte Spannungsmessgerät zunächst an Schaltungen zu testen, wo bereits mit einem Multimeter Spannungen gemessen wurden, um Vergleichswerte zur Verfügung zu haben

(Wechsel zu nächster Folie, die dieselbe wie die vorangegangene Folie ist)

Inhaltsfolie 4 zur Spannungsmessung vgl. Inhaltsfolie 2 zur Spannungsmessung:

Inhaltsfolie 1 zur Widerstandsmessung:



Textvorschlag:

...dabei werden in elektrischen Stromkreisen (sogenannte Ohm'sche) Widerstände bewusst eingebaut, um den Elektrizitätsstrom zu begrenzen (und damit andere Bauteile vor zu großem Elektrizitätsstrom zu schützen). Dazu benötigt man unterschiedlich große Widerstandswerte (mit der Einheit Ohm, Ω), sodass es zweckmäßig ist, den jeweiligen Widerstandswert auf den Bauteilen – durch farbige Ringe codiert – ablesen zu können.

Auf der Folie siehst du sechs Widerstand-Bauteile und die entsprechende Decodierungstabelle – die Bauteile haben demnach die Widerstandswerte:

(der Reihe nach durchgehen und besprechen):

„links oben": rot – rot – braun: 2 2 mal 10 = 220 Ω

„rechts oben": orange – orange – braun: 3 3 mal 10 = 330 Ω

„links mitte": gelb – violett – braun: 4 7 mal 10 = 470 Ω

„rechts mitte": blau – grau – braun: 6 8 mal 10 = 680 Ω

„links unten": braun – schwarz – rot: 1 0 mal 100 = 1000 Ω = 1 kΩ (Kilo-Ohm)

„rechts unten": braun – schwarz – grün: 1 0 mal 100000 = 1000000 = 1 MΩ (Meg-Ohm)

Inhaltsfolie 2 zur Widerstandsmessung:

Textvorschlag:

Mitunter ist es einfacher, den Widerstandswert eines Bauteils mit dem Multimeter zu messen, als diesen Wert mit Hilfe der farbigen Ringe und einer Decodierungstabelle zu bestimmen:

Dazu wählst du am besten Messfühler, die mit kleinen Häkchen die Beine des Bauteils „umklammern" können, verbindest sie mit dem Bauteil und stellst am Multimeter einen passenden Messbereich für den Widerstand ein (evtl. auf Folie den Messbereich für die Widerstandsmessung aktiv zeigen).

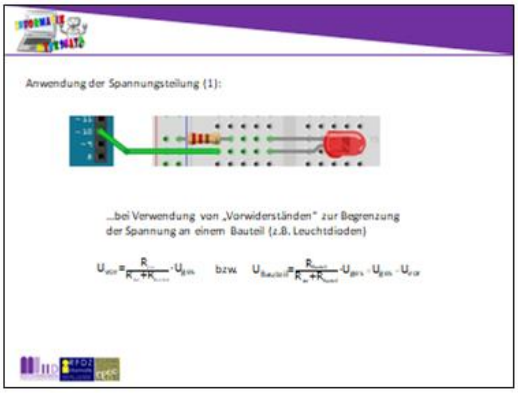
(geringfügige) Abweichungen vom Widerstandswert, der sich durch Decodieren der farbigen Ringe ergibt, sind häufig – der vierte farbige Ring am Bauteil gibt ja an, um wie viel Prozent der Widerstandswert vom codierten Sollwert abweichen darf.

Allerdings können Abweichungen des gemessenen Wertes von diesem Sollwert auch darauf hindeuten, dass die Batterie des Multimeters gewechselt werden sollte.

(beim Widerstandsbauteil auf dem Photo ist der vierte Ring silber, die Toleranz beträgt daher 10%: Der Widerstandswert kann im Bereich von 217,8 Ohm bis 222,2 Ohm liegen – der angezeigte Messwert muss daher nicht unbedingt auf Erschöpfung der Multimeterbatterie hindeuten).

Als Beispiele für die Anwendung (Ohm'scher) Widerstände werden in der Informationsdatei **ST_I_12 Elektrizität_Praxis_Spannung_Widerstand** vier Spannungsteiler-Schaltungen vorgestellt, die Ausgangspunkte für weitere Überlegungen im Rahmen der Arbeitsanregungen aus der Datei **ST_AA_12 Elektrizität_Praxis_Spannung_Widerstand** dienen:

Inhaltsfolie 1 zu „Anwendung der Spannungsteilung":

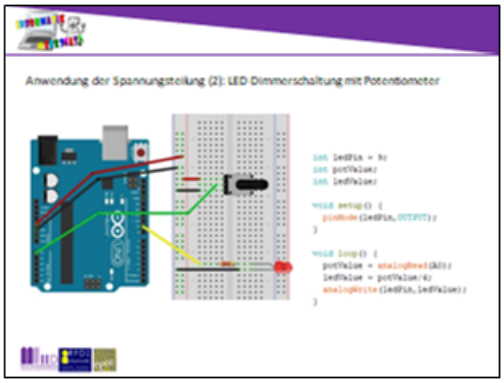


Textvorschlag:

...Text von Folie vorlesen, allenfalls folgende Erklärung anschließen:

Durch die Verwendung eines Vorwiderstandes soll erreicht werden, dass ein Teil der insgesamt zur Verfügung stehenden Spannung Uges an diesem Vorwiderstand anliegt, sodass am (zumeist empfindlichen) Bauteil nur noch die restliche Spannung anliegt. Je nach Wahl des Vorwiderstandes kann für den Bauteil unterschiedlich hohe Spannung (bis höchstens Uges) bereitgestellt werden.

Inhaltsfolie 2 zu „Anwendung der Spannungsteilung":



Textvorschlag:

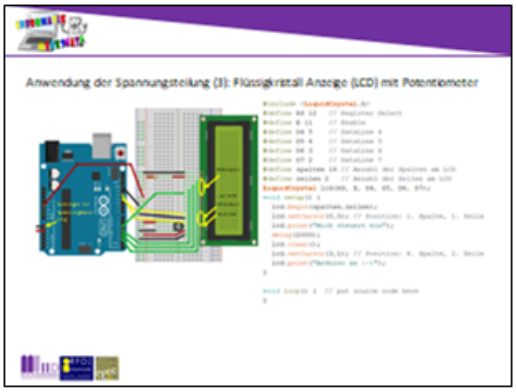
Die Schaltung sollte selbsterklärend sein, zum Programmcode ist allerdings anzumerken:

Mit analogWrite kann auch über einen mit „~" gekennzeichneten digitalen Pin ein analoges Signal „simuliert" werden. Dazu wird das digitale Signal während einer Zeitspanne widerholt ein- und wieder ausgeschaltet. Wie oft das Ein- und Ausschalten passiert, wird durch den zweiten Parameter des analogWrite-Befehls gesteuert. Dieser Parameter kann eine ganze Zahl von 0 bis 255 sein: Je kleiner der Parameterwert ist, desto länger sind das Signal zwischendurch ausgeschaltet. Wird ein solches Signal an eine Leuchtdiode angelegt, scheint diese dem menschlichen Auge mit nur geringer Intensität zu leuchten. Bei einem hohen Parameterwert, werden die „Signalpausen" kürzer, die Leuchtdiode scheint heller zu leuchten.

Aus dem Wertebereich für den zweiten Parameter des analogWrite-Befehls erklärt sich auch die Umrechnung zwischen dem Wert der Variablen potValue (0 bis 1023) und der Variablen ledValue (0 bis 255) – der zweite Wert kann einfach als ein Viertel des ersten Werts erhalten werden... (evtl. die jeweils passenden Stellen auf der Folie zeigen)

Anmerkung: Je nach Lerngruppe kann diese Spannungsteilerschaltung und das zugehörige Programm auch als einführende Aufgabe zu den Arbeitsanregungen mit Spannungsteilern gestellt werden.

Inhaltsfolie 3 zu „Anwendung der Spannungsteilung“:



Textvorschlag:

Bei der Flüssigkristall-Anzeige (engl.: Liquid Crystal Display, LCD) dient das Potentiometer nur dazu, die Helligkeit der Anzeige zu regeln. Dennoch ist dieser Bauteil nicht unbedeutend, gelingt es doch mit ihm, ein von einem Computer unabhängiges Arduino-Voltmeter zusammenzustecken und zu programmieren.

Programmtechnisch ist zu bemerken, dass
-mit `LiquidCrystal` erstmals eine vordefinierte (d.h. nicht-eigene) Arduino-Programmbibliothek verwendet wird;
-mit dem Präprozessor-Befehl `#define` das selbe erreicht wird wie bei einer Variablendeklaration mit gleichzeitiger Wertzuweisung;
-mit `LiquidCrystal lcd(RS, E, D4, D5, D6, D7)` eine sogenannte Objektvariable `lcd` vereinbart wird, über die der LCD-Bauteil programmiert werden kann.
-mit `lcd.begin(spalten, zeilen)` die Variable `lcd` mit der gewünschten Anzahl Anzahl von Spalten und Zeilen initialisiert wird;
-mit `lcd.setCursor(spalte, zeile)` eine Ausgabeposition am LCD gewählt wird, wobei die **Zählung der Spalten und Zeilen bei Null beginnt!**

Inhaltsfolie 4 zu „Anwendung der Spannungsteilung“:

Textvorschlag:

Für die vierte beispielhafte Anwendung einer Spannungsteiler-Schaltung wird bewusst auf die aus Abschnitt 9 bekannte Steuerung eines Piezo-Lautsprechers zurückgegriffen, da durch die gestellten Arbeitsanregungen in der Aufgabendatei ein vertieftes Verständnis der Spannungsteilung erzielt werden soll.

Gleichzeitig wird bei dem angebotenen Beispielcode mit `map` ein neuer Befehl vorgestellt, der es gestattet, einen innerhalb eines vorgegebenen Bereichs gemessenen Spannungswert „direkt“ in den entsprechenden Wert eines analogen Ausgangssignals umzurechnen.

…siehe auch nächste Seite

Textvorschlag:

Mit dem angegebenen `map` -Befehl wird der durch die Kalibrierung des Photowiderstandes festgelegte Spannungsbereich zwischen den in den Variablen `sensorLow` und `sensorHigh` gespeicherten Werten auf den „Zielbereich" für die Frequenz, hier von 50 bis 4000 Hertz), abgebildet und die relative Position des in der Variablen `sensorValue` gespeicherten „Messwertes" von [`sensorLow`, `sensorHigh`] auf [50, 4000] umgerechnet.

Dies bedeutet: Der Abstand (`sensorValue` – `sensorLow`) verhält sich zur Breite des Spannungsbereichs (`sensorHigh` – `sensorLow`) genauso, wie der Abstand der in der Variablen namens `pitch` gespeicherte Wert der entsprechenden Tonhöhe von der Untergrenze des Frequenzintervalls, (`pitch` – 50), zur Breite des Frequenzintervalls, (4000 – 50), bzw. als Formel:

$$(\texttt{sensorValue} - \texttt{sensorLow}) : (\texttt{sensorHigh} - \texttt{sensorLow}) = \\ = (\texttt{pitch} - 50) : (4000 - 50)$$

...woraus (leicht?) eine Formel zur Berechnung von `pitch` umgestellt werden kann...