# Arduino und Programmieren: Grundlagen

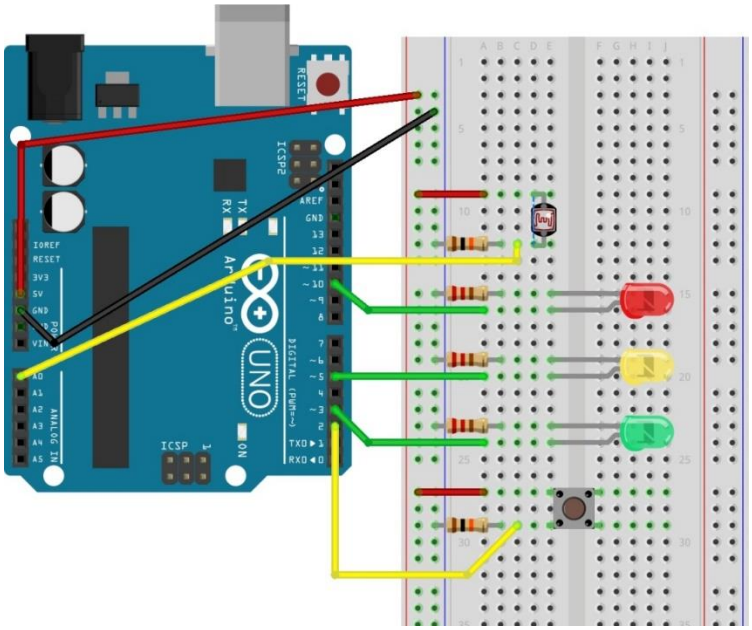
Aufbauend auf dem Wissen um die Bedeutung geschlossener Stromkreise werden im zweiten Teil des Arbeitspaktes zu Schaltungstechnik und Programmieren mit dem Arduino-Board grundlegende Konzepte des Programmierens dargeboten. Dies sind:

- das Konzept der **Variablen** als Möglichkeit zum „Merken“ einzelner Daten innerhalb eines Programms,
- das Konzept der **Programmsequenz** als schrittweise abzuarbeitende Folge von Programmbefehlen,
- das Konzept von **Unterprogrammen**, mit denen eigene Arduino-Befehle programmiert werden können, durch deren Verwendung der Programmcode übersichtlicher gehalten werden kann,
- das Konzept der Programmschleife, die es gestattet, eine Folge von Befehlen mehrmals hintereinander auszuführen, und
- das Konzept der **Programmverzweigung**, mit der nach Überprüfung einer Bedingung ein Programmteil ausgeführt oder übersprungen werden kann.

Jedem dieser vier Konzepte ist ein eigenes Arbeitsblatt gewidmet. Diese Arbeitsblätter – samt zugehörigen Informationsdateien bauen aufeinander auf und sollten in der Reihenfolge ihrer Nummerierung (ST_AA05… bis ST_AA08) bearbeitet werden.

# Arbeiten in der Arduino-Programmierumgebung

Der schaltungstechnische Kern dieses Teils des Arbeitspakets ist die gemäß nachstehender Abbildung vorbereitete Schaltung, die zum Programmieren einer Ampelsteuerung oder eines optischen Morseapparats verwendet wird:



Anhand der aus Aufgabenblatt **ST_AA04Arduino_erstes_Programm** bekannten Programme

```
void setup() {
  pinMode(5,OUTPUT);
  digitalWrite(5,LOW);
}

void loop() {
  digitalWrite(5,HIGH);
  delay(2000);
  digitalWrite(5,LOW);
  delay(5000);

}
```

```
int ledPin = 5;

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

void loop() {
  digitalWrite(ledPin,HIGH);
  delay(2000);
  digitalWrite(ledPin,LOW);
  delay(5000);

}
```

ist zunächst der prinzipielle Aufbau eines Arduino-Programms in der gewählten Programmierumgebung hervorzuheben:

Jedes **Arduino-Programm** besteht aus **zumindest zwei** **Programmteilen**.

Diese Programmteile haben die Namen `setup()` bzw `loop()`, wobei der **Code** jedes dieser **Programmteile** von einem **Paar geschwungener Klammern**, **{ }**, **eingeschlossen** wird.

Dabei wird

der `setup()`-**Programmteil** **nur einmal** zu Programmstart ausgeführt,

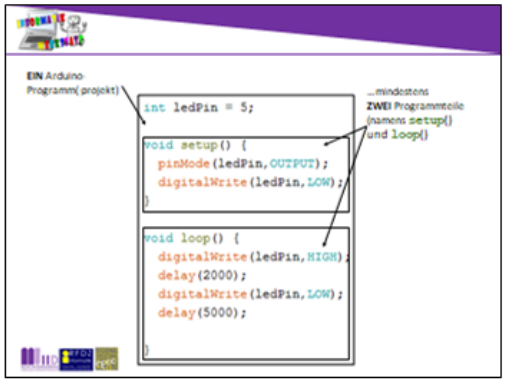
der `loop()`-**Programmteil** hingegen **immer wieder ausgeführt**, solange das Arduino-Board mit Spannung versorgt wird.

Dies ist einerseits wichtig, weil erst mit diesem Grundwissen die Wirkung eines Arduino-Programms auf die programmierte Schaltung verstanden werden kann, und andererseits, weil dadurch die Basis für das Programmieren eigener Arduino-Befehle (= weitere Programmteile) gelegt wird.

Zusätzlich sollten in einem ersten Schritt die in den Programmen verwendeten Befehle erklärt werden.

Obwohl diese Informationen in der Informationsdatei „**ST_I_Arduino_Programmierumgebung**“ dargeboten werden, empfiehlt sich ein ergänzender Vortrag für alle unter Verwendung der dem Arbeitspaket beigefügte Kurzpräsentation namens „**ST_PR_Arduino_Programmierumgebung.pptx**“:

Inhaltsfolie 1:

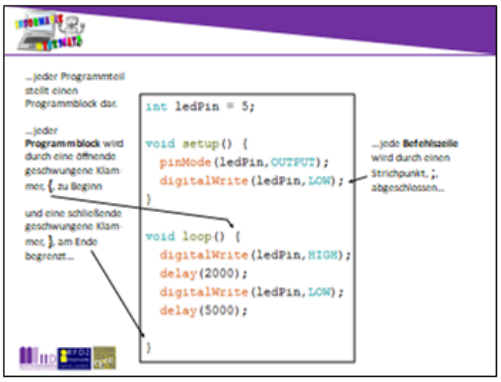



Textvorschlag:

Im `setup()`–Programmteil werden die Anfangseinstellungen (z.B. für die Leuchtdioden) festgelegt. Dieser Programmteil wird einmal, gleich nach dem Programmstart ausgeführt.

Im `loop()`-Programmteil können die Einstellungen fortwährend geändert werden, solange das Arduino-Board mit Spannung versorgt wird. Beispielsweise können die Leuchtdioden ein- bzw. ausgeschaltet werden.

Inhaltsfolie 2:




Hinweis:
Zur Wahrung der Übersichtlichkeit sind jeweils nur eine öffnende bzw. schließende geschwungene Klammer und ein Strichpunkt durch die eingezeichneten Pfeile hervorgehoben. Bei Präsentation dieser Folie sollten aber ALLE geschwungenen Klammern und Strichpunkte (z.B. mit Hilfe eines Lichtzeigers oder eines Zeigestabes) gezeigt werden!

Inhaltsfolie 3:


Textvorschlag:
Mit dem Befehl

**pinMode** wird festgelegt, ob über den Steckkontakt mit der angegebenen Nummer (hier: 5) Spannung zur Verfügung gestellt werden kann (Modus: **OUTPUT**), oder ob über diesen Steckkontakt Spannung gemessen werden kann (Modus: **INPUT**).

**digitalWrite** wird für einen im Modus **OUTPUT** betriebenen Steckkontakt mit der angegebenen Nummer (hier: 5) festgelegt, ob die Spannung 5 Volt (**HIGH**) oder die Spannung 0 Volt (**LOW**) über diesen Steckkontakt zur Verfügung gestellt wird. Mit anderen Worten: Der an diesem Steckkontakt angeschlossene Bauteil wird entweder eingeschaltet (**HIGH**) oder ausgeschaltet (**LOW**).

**delay** unterbricht die Programmausführung für die angegebene Anzahl von Millisekunden (2000 Millisekunden entsprechen zwei Sekunden)

(Tastendruck zum Einblenden der Variablendeklaration)

Für die Nummer des Steckkontakts kann auch eine **Variable** verwendet werden. Eine Variable vom Datentyp int kann eine ganze Zahl speichern.

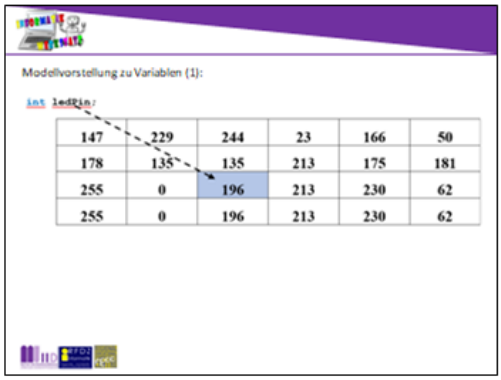
Damit sind sämtliche Basisinformationen verfügbar, damit die ersten sechs Aufgaben in der Datei „**ST_AA_05Arduino_und_Programmierumgebung**“ bearbeitet werden können. Aufgabe 1) bis Aufgabe 3) sollen Programmiergrunderfahrungen vermitteln, indem die bereitgestellte Schaltung als Ampel programmiert und gesteuert wird. In Aufgabe 4) bis Aufgabe 6) werden diese Grunderfahrungen weiter geübt, indem die Schaltung als optischer Morseapparat programmiert wird. Allfällig benötigte Zusatzinformationen und vorbereitende Übungen zum Morsecode sind in den beigefügten Dateien „**CO_I_Einführung**“ bzw. „**CO_AA_Morsecode**“ (aus dem Werkstatt-Arbeitspaket zur Codierung) zu finden.

Wesentlich ist zudem die frühe Entwicklung einer tragfähigen Modellvorstellung zum Variablenkonzept:

Eine **Variable** gibt einem **Bereich im Speicher des Computers** einen **Namen**. Durch Verwendung dieses Namens im Computerprogramm kann auf den Speicherbereich, auf den die Variable **verweist**, zugegriffen werden.

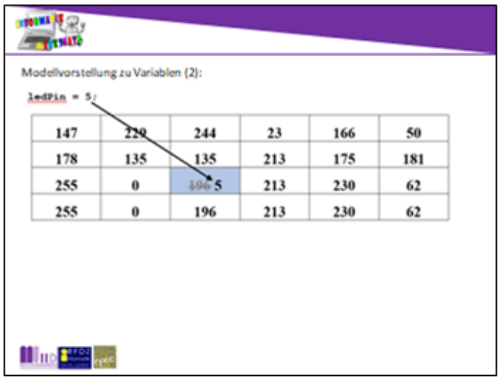
In der Informationsdatei „**ST_I_05Arduino_Programmierumgebung**“ wird die zugehörige Modellvorstellung bildhaft entwickelt, wobei auch hier die individuelle Erarbeitung durch Besprechung der relevanten Folien aus „**ST_PR_Arduino_Programmierumgebung.pptx**“ unterstützt werden sollte:

Inhaltsfolie 4:



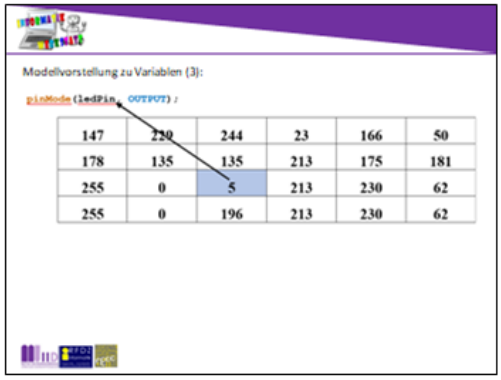

Textvorschlag:
Bei der Vereinbarung einer Variablen wird ein Speicherbereich passender Größe (abhängig vom Datentyp der Variablen – hier: int) reserviert. In diesem Speicherbereich ist zunächst irgendein Wert gespeichert....

Inhaltsfolie 5:




Textvorschlag:
Bei der Wertzuweisung an eine zuvor deklarierte Variable wird der zugewiesene Wert in die reservierte Speicherzelle geschrieben. Der Wert, der zuvor in dieser Speicherzelle gespeichert war, geht verloren!

Inhaltsfolie 6:




Textvorschlag:
Wenn die Variable verwendet wird, wird der in der reservierten Speicherzelle gespeicherte Wert gelesen. Dabei wird der gespeicherte Wert NICHT verändert!

Anzumerken ist, dass der Speicher im Computer tatsächlich linear (d.h. „eindimensional“) verwaltet wird, was als **eine** „Zeile“ (oder „Spalte“) aufeinanderfolgender Speicherbereiche vorgestellt werden kann. Die Darstellung des Speichers als zweidimensionale „Speichertabelle“ ist allerdings platzeffizienter und vermittelt eine auf dieser Stufe ebenso korrekte Modellvorstellung für die Verwendung von Variablen.

Zur Festigung dieser Modellvorstellung dienen Aufgabe 7 bis Aufgabe 9 der Aufgabendatei „**ST_AA_05Arduin _Programmierumgebung**“.

# Arduino – eigene Befehle programmieren

Das Programmieren eigener Befehle für das Arduino-Board bedeutet, zusätzlich zum `setup`()- bzw `loop`()-Programmteil weitere Programmteile zu codieren. Jeder dieser Programmteile hat (bis auf Weiteres) die folgende Struktur:

```
void programmTeilName(Daten, die der neue Befehl benötigt) {
      programmBefehl1;
      programmBefehl2;
      …
}
```

Jeder zusätzlich codierte Programmteil benötigt demnach (bis auf Weiteres)

- einen **Namen**, der im Wesentlichen frei gewählt werden kann. Beispiele für vordefinierte Namen von Programmteilen sind `setup` oder `loop` (diese dürfen nicht noch einmal verwendet werden). Wird als Name des Programmteils ein zusammengesetztes Wort gewählt, gehört es zum guten Programmierstil, das erste (Teil-) Wort klein, die anderen groß zu beginnen, beispielsweise eben `programmTeilName`. Dies verbessert die Lesbarkeit des Codes;
- das **Schlüsselwort** `void` VOR dem Programmteilnamen;
- ein Paar runder Klammern, **()**, die die Daten beinhalten, die der Befehl (d.h. der Programmteil) benötigt – werden keine Daten benötigt, **darf** dieses Klammerpaar auch **leer bleiben**, es **muss aber** jedenfalls **hingeschrieben werden**!

  Wenn der selbst codierte Befehl Daten „von außen“ benötigt, sind diese (bis auf Weiteres) im Befehlscode in der Form

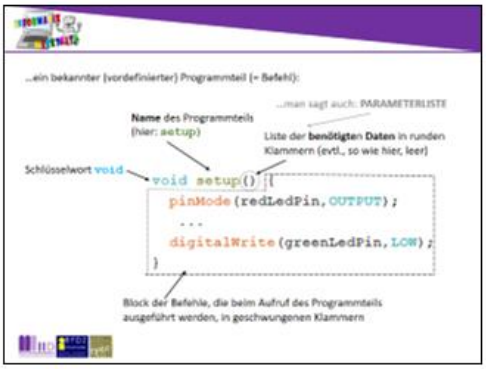
  ```
  …programmTeilName(int erstesDatum, int zweitesDatum,…)
  ```

  anzugeben.

  Das Schlüsselwort `int` zeigt dabei an, dass das dem Befehl „mitgegebene“ Datum (auch als Parameter bezeichnet) eine ganze Zahl ist. Die Schreibweise für diese „mitgegebenen“ Parameter gleicht nicht zufällig der bekannten Vereinbarung von Variablen Parameter können im Befehls block (s.u.) des neuen Programmteils wie Variable verwendet werden;
- den in geschwungene Klammern, **{}**, eingeschlossenen Befehlsblock, der alle vorcodierten (z.B. `delay`, `digitalWrite`,…) oder bereits selbst codierten (Programm-) Befehle beinhalten kann. Es ist wiederum darauf zu achten, dass jede **Befehlszeile** mit einem **Strichpunkt**, **;**, **abgeschlossen werden muss**.

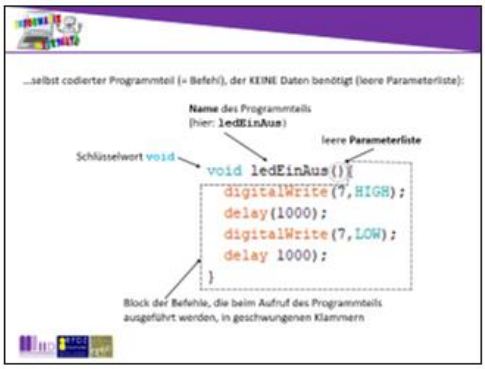
Beispiele für selbst codierte Befehle und wie diese verwendet werden können finden sich in der Informationsdatei „**ST_I_06Arduino_Modularisierung**“, sowie in der Präsentation namens „**ST_PR_Arduino_Modularisierung.pptx**“, deren Folien zusätzlich als Basisinformation gezeigt und erläutert werden können (siehe nächste Seite).

Inhaltsfolie 1:



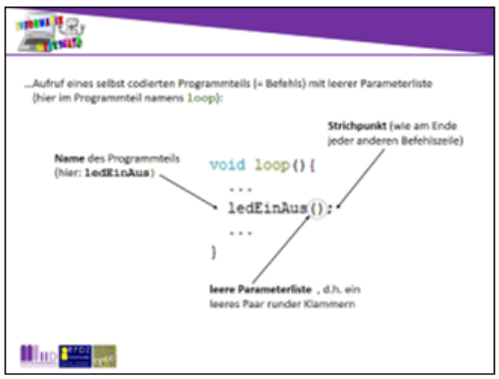

Hinweis:
Die Einblendung der Textfelder erfolgt jeweils nach Drücken der <Enter>-Taste.
Zusätzlich zur visuellen Repräsentation sollten die Bestandteile eines Programmteils sinnvollerweise auch beSPROCHEN werden

Inhaltsfolie 2:



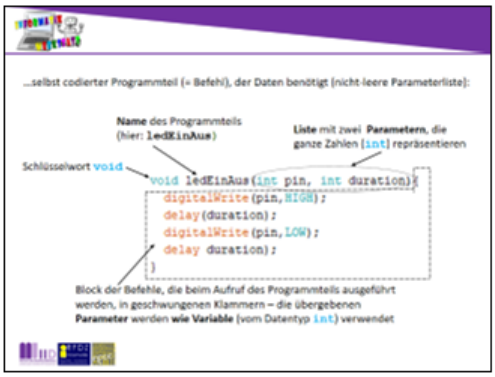

Textvorschlag:
Selbst codierte Programmteile sind gleich wie die bereits bekannten Programmteile **setup** und **loop** aufgebaut.
Der Name eines selbst codierten Programmteils kann frei gewählt werden – es gehört zu guter Programmierpraxis, bei Verwendung eines zusammengesetzten Wortes als Programmname, ab dem zweiten Wortteil den Anfangsbuchstaben des jeweiligen Wortteils groß zu schreiben.
Zu beachten ist auch die Einrückung des Programmcodes gegenüber der „Überschriftenzeile" und der schließenden geschwungenen Klammer (dies evtl. auch auf der projizierten Folie zeigen)

Inhaltsfolie 3:



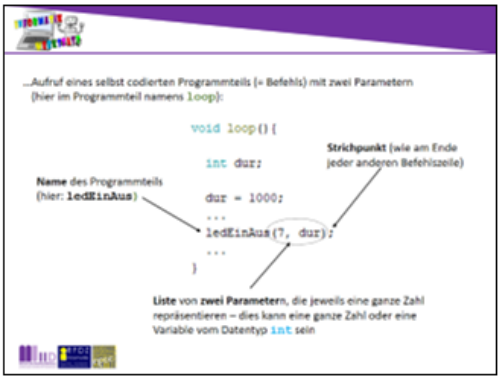

Textvorschlag:
Wie die vordefinierten Befehle werden auch selbst codierte Befehle durch Angabe ihres Namens und der Parameterliste aufgerufen (auf den Strichpunkt am Ende der Befehlszeile darf durchaus wieder verwiesen werden).

Inhaltsfolie 4:




Textvorschlag:
Wenn die Parameterliste nicht leer ist, sind beim Codieren eines neuen Befehls sämtliche Parameter mit ihrem Datentyp (hier: **int**) und ihrem Namen (in dieser Reihenfolge) in der durch die runden Klammern begrenzten Parameterliste anzugeben und durch Beistrich voneinander zu trennen.
Der Datentyp **int** zeigt an, dass ein Parameter eine ganze Zahl repräsentiert (speichert).
Die Liste der Parameter im Befehlscode ähnelt sehr der Deklaration von Variablen – die Namen der Parameter können im Codeblock des neuen Befehl auch wie Variable verwendet werden.

Inhaltsfolie 5:




Textvorschlag:
Beim Aufruf eines selbst codierten Befehls müssen für die Parameter Werte des passenden Datentyps (hier: **int** – ganze Zahlen) in die durch die runden Klammern begrenzte Parameterliste geschrieben werden. Dabei kann der ganzzahlige Parameterwert entweder direkt als ganze Zahl oder als Variable vom Datentyp **int** angegeben werden.

Hinweis: Programmteile mit Rückgabewert (d.h. solche, bei denen vor dem Namen nicht `void` sondern z.B. `int` oder `float` steht), sowie solche, die nicht-ganzzahlige Parameterwerte erfordern, werden in späteren Teilen dieser Arbeitspakte thematisiert. Für den jetzigen Kenntnisstand der Lernenden genügt die hier präsentierte „vereinfachte“ Struktur, um das informatisch wichtige Konzept der **Modularisierung** aktiv kennen zu lernen.

Arbeitsaufträge zum Einüben und zum Vertiefen des Codierens eigener Programmteile (Befehle) finden sich in der Datei „**ST_AA_06Arduino_Modularisierung**“. Die ersten drei Arbeitsaufträge betreffen wieder die Programmierung der bekannten Schaltung als dreifärbige Ampel, Arbeitsaufträge 4 bis 6 beziehen sich auf das optische Morsen mit dieser Schaltung.

# Arduino – Wiederholungen effizient programmieren

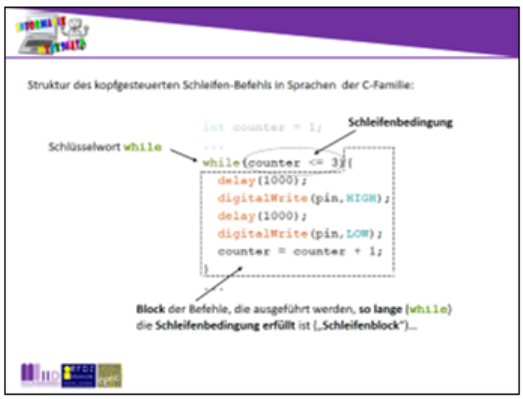
Der Abschnitt dieses Arbeitspakets zu Wiederholungen in Arduino-Programm(teil)en („Programmschleifen“) wird motiviert durch die Aufgabenlösungen zum mehrmaligen Blinken einer Leuchtdiode aus dem vorangegangenen Abschnitt zur Modularisierung (vgl. Informationsdatei „**ST_I_07Arduino_Wiederholungen**“). Dabei wird aus didaktischen Gründen nur der Typ der **kopfgesteuerten Schleife** verwendet, die in Programmiersprachen der C-Sprachfamilie in der Form

```
while(Schleifenbedingung) {
      // Block mit Befehlen, die
      // ausgeführt werden, solange
      // die Schleifenbedingung
      // WAHR ist
}
```

formuliert wird. Die fachdidaktische Forschung zeigt, dass das parallele Erlernen mehrerer (aller) in einer Programmiersprache vorhandenen Schleifentypen, zu Beginn eher verwirrend denn hilfreich ist.

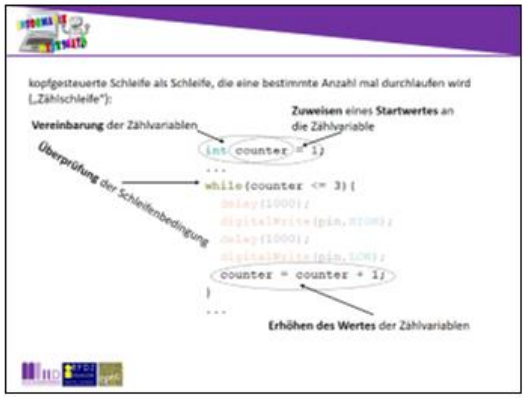
Zudem wird die kopfgesteuerte Schleife zunächst als Zählschleife verwendet – zur Erklärung deren „Funktionsweise“ dienen die Folien der Datei „**ST_PR_Arduino_Wiederholungen.pptx**“:

Inhaltsfolie 1:




Hinweis:
Diese Folie zeigt die allgemeinen Charakteristika der kopfgesteuerten Schleife – es empfiehlt sich, die mit Pfeilen beigefügten Erläuterungen zu besprechen, damit grundlegende Begrifflichkeiten eingeprägt werden.

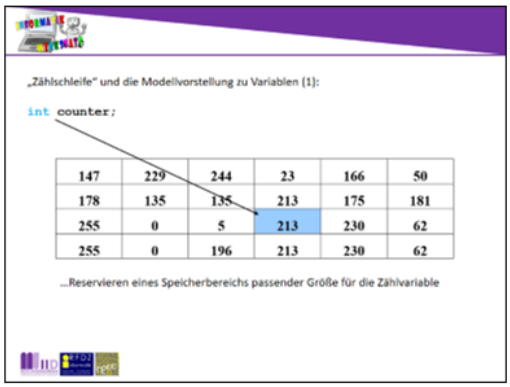
Inhaltsfolie 2:




Hinweis:
Zunächst wird die kopfgesteuerte Schleife als Zählschleife formuliert – auch diese Folie sollte besprochen werden, damit die betreffenden Begrifflichkeiten bekannt werden

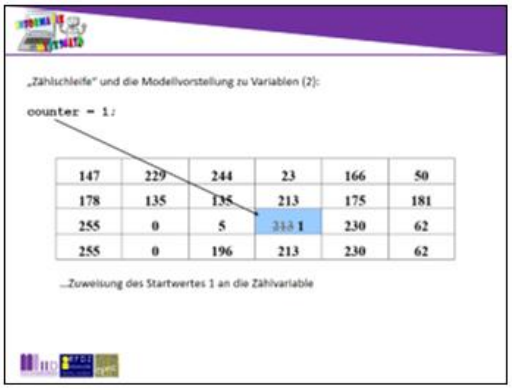
Hinweis: Da in Aufgabe 2 der Aufgabendatei „**ST_AA_07Arduino_Wiederholungen**“ auf die Modellvorstellung zu Variablen im Zusammenhang mit Zählschleifen verwiesen wird, sollten die Inhaltsfolien 3 bis 14 jedenfalls gezeigt und besprochen werden.

Inhaltsfolie 3:

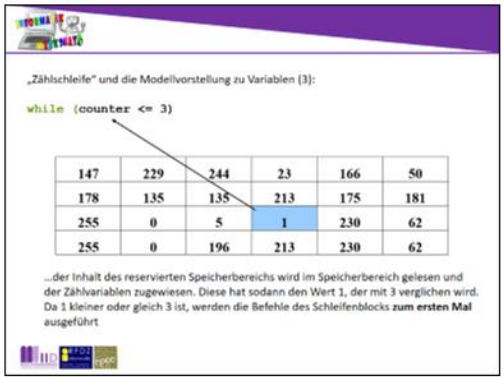



Hinweis:
Diese und die folgenden Folien dienen dazu, den etwas abstrakten Prozess des Weiterzählens innerhalb einer Zählschleife zu visualisieren. Dazu eignet sich die Modellvorstellung von Variablen. Obwohl der Begleittext auf den Folien jeweils vermerkt ist, sollte er besprochen („vorgelesen“) werden. Dies meint insbesondere die wiederkehrenden gleichen Formulierungen, durch die die „Natur“ der Programmschleife sichtbar wird.

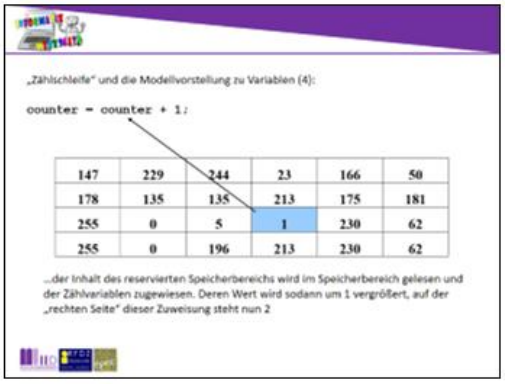
Inhaltsfolie 4:



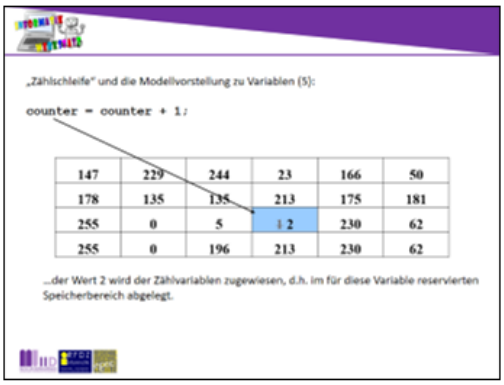

Inhaltsfolie 5:



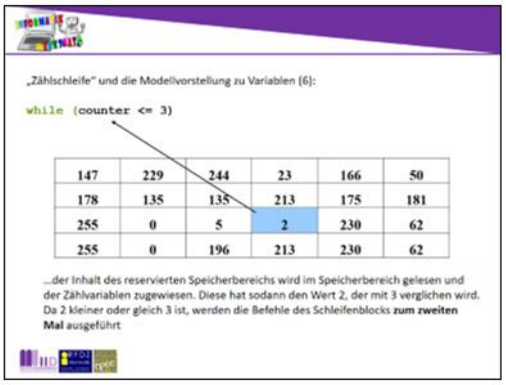

Inhaltsfolie 6:



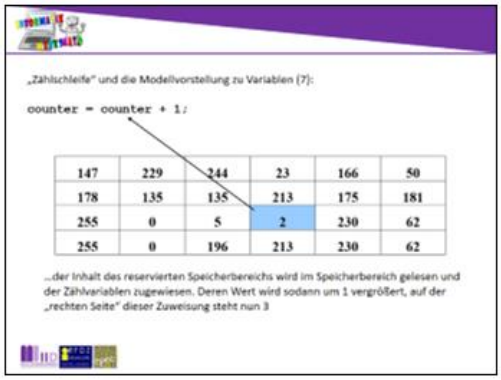
Inhaltsfolie 7:



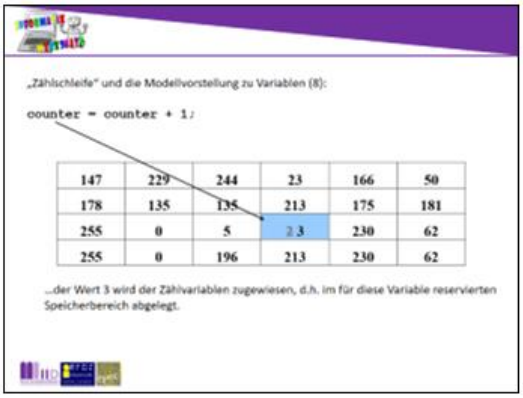

Inhaltsfolie 8:



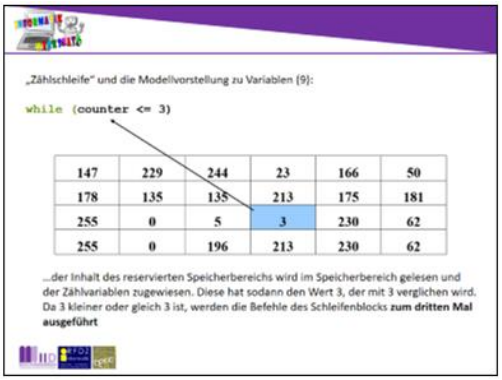

Inhaltsfolie 9:



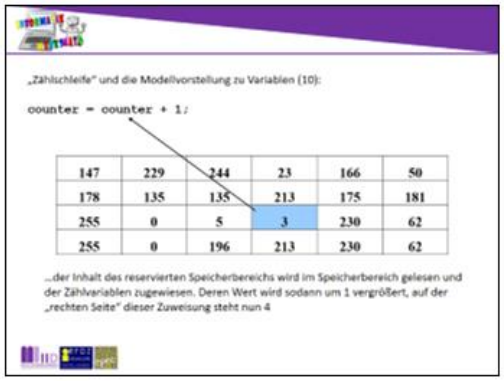

Inhaltsfolie 10:



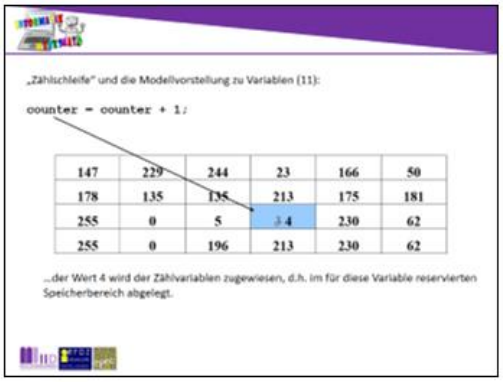
Inhaltsfolie 11:



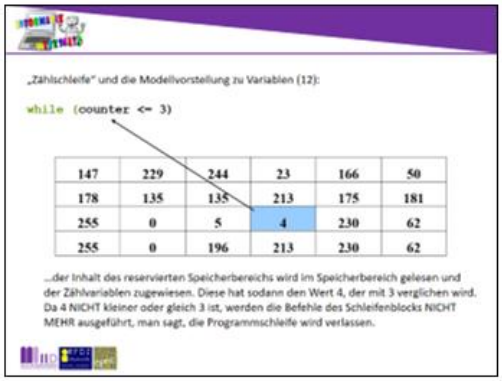

Inhaltsfolie 12:




Inhaltsfolie 13:




Inhaltsfolie 14:



# Arduino – ein Programm trifft Entscheidungen

Im letzten Abschnitt des Arbeitspakets zu Grundlagen der Programmierung mit dem Arduino-Board wird durch die Verwendung der zusätzlichen Schaltelemente der vorbereiteten Schaltung, den Druckknopf-Schalter und den Photowiderstand, das Codieren von Programmverzweigungen motiviert:

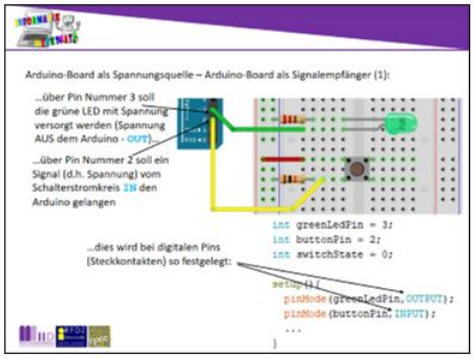
Über das auf das Arduino-Board geladene Programm soll überprüft werden,

- ob (`if`) der Schalter gedrückt wurde (und eine entsprechende Aktion veranlasst werden), bzw.
- ob (`if`) ausreichend Licht auf den Photowiderstand fällt (und eine entsprechende Aktion veranlasst werden) oder nicht (und dann ebenfalls eine für diese Situation passende Aktion veranlasst werden).

Da die Signalübermittlung von Druckknopf-Schalter bzw. Photowiderstand sowohl hinsichtlich der Schaltung als auch hinsichtlich des Programmcodes unterschiedlich realisiert wird, teilt sich dieser Abschnitt in zwei Teile:

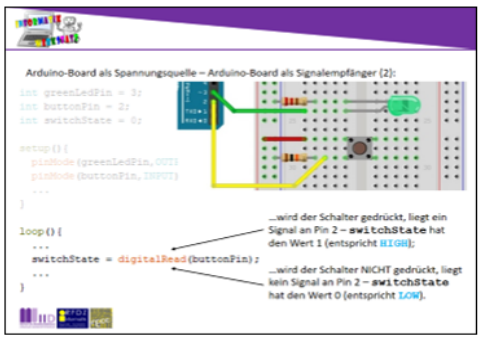
### Verarbeitung des Druckknopf-Schalter-Signals mit einer einseitigen Programmverzweigung:

Der Druckknopf-Schalter kennt zwei Zustände: „Schalter gedrückt“ bzw. „Schalter nicht gedrückt“. Dies kann durch zwei Spannungswerte (Spannungpegel hoch – `HIGH` – bzw. Spannungspegel niedrig – `LOW`) codiert werden. Daher kann das **Schaltersignal über einen digitalen Steckkontakt** an das Arduino-Board übermittelt werden: Über einen digitalen Steckkontakt kann ja bekanntlich ein Bauteil mit Spannung versorgt werden (oder eben nicht) – genauso kann über einen digitalen Steckkontakt auch festgestellt werden, ob an ihm Spannung anliegt (oder eben nicht). Im Programm muss nur festgelegt werden, dass der digitale Steckkontakt im **Modus** `INPUT` (und nicht wie bisher im **Modus** `OUTPUT`) betrieben werden soll. Details dazu finden sich auf den ersten beiden Inhaltsfolien der Datei „**ST_PR_Arduino_Verzweigungen.pptx**“, die durchaus wieder ergänzend zu den Erläuterungen in der Informationsdatei namens „**ST_I_08Arduino_Verzweigungen**“ gezeigt (und besprochen) werden sollen:




Textvorschlag:

Bisher dienten die **digitalen Steckkontakte** (Pins) des Arduino-Boards als Spannungsquelle für die programmierbare Schaltung. Über die digitalen Pins können aber auch Signale vom Arduino-Board registriert werden – z.B. wenn ein Schalter betätigt wurde...
(der weitere Text entspricht jenem auf der Folie)




Textvorschlag:

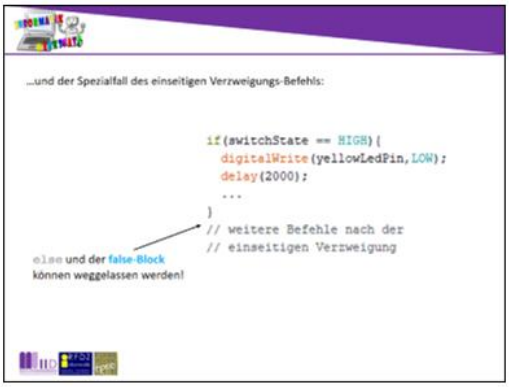
Wenn an einem digitalen Steckkontakt im Modus INPUT ein Signal anliegt, bedeutet das (hohe) Spannung (**HIGH** = 1) an diesem Steckkontakt, liegt andererseits jedoch kein Signal an, bedeutet dies, dass am Steckkontakt keine Spannung anliegt (**LOW** = 0).
(es sollte erwähnt werden, dass die Schlüsselwörter **HIGH** und **LOW** lediglich Platzhalter für die ganzen Zahlen 1 und 0 sind....).
Programmtechnisch ist es wünschenswert, entscheiden zu können, ob der Schalter gedrückt wurde oder nicht, d.h. ob am digitalen Pin mit der Nummer 2 ein Signal anliegt oder nicht.
Diese Überleitung motiviert die Kontrollstruktur „Programmverzweigung“ (s. nächste Folie)

Bei der Vorstellung des Codes für den „Verzweigungs-Befehl“ zum Treffen von programmgesteuerten Entscheidungen sollte zunächst das Codegerüst für die vollständige (**zweiseitige**) **Verzweigung** erläutert werden:

```
if(Bedingung){
      // Befehle, die ausgeführt werden sollen,
      // wenn die Bedingung erfüllt (true) ist
      // („true-Block“
}
else{
      // Befehle, die ausgeführt werden sollen,
      // wenn die Bedingung NICHT erfüllt (false) ist
      // („false-Block“
}
```

Aufgrund der gewählten Arbeitsanregungen zur Schalterprogrammierung in der Aufgabendatei namens „**ST_I_08Arduino_Verzweigungen**“ wird zunächst allerdings nur die einseitige Verzweigung (ohne `else` und „`false`-Block“) benötigt. Dem wird auch auf den Inhaltsfolien 3 und 4 Rechnung getragen:


Hinweis:
Diese Folie zeigt die allgemeine Struktur einer zweiseitigen Verzweigung – es empfiehlt sich, die mit Pfeilen beigefügten Erläuterungen zu besprechen, damit grundlegende Begrifflichkeiten eingeprägt werden, und die Idee hinter diesem Befehl zu erläutern:
Überprüfe, ob (**if**) ein Signal am Pin Nummer 2 anliegt: Wenn das der Fall ist .... (schalte die gelbe Leuchtdiode aus und warte zwei Sekunden), ansonsten (**else**) ... (schalte die rote Leuchtdiode aus und warte zwei Sekunden).
Es wird bei der Verzweigung also geprüft, ob die angegebene Bedingung erfüllt (wahr, **true**) ist oder nicht erfüllt (falsch, **false**) ist: Wenn die Bedingung erfüllt (**true**) ist, wird der auf die Zeile mit **if** folgende Befehlsblock (**true**-Block) ausgeführt, ansonsten der auf die Zeile mit **else** folgende Befehlsblock (**false**-Block).
Der Codeteil ab dem Schlüsselwort **else** kann – wenn nicht benötigt – auch weggelassen werden (dann spricht man von einer einseitigen Verzweigung). Daher ist dieser Codeteil auf der Folie auch mit geringerer Schriftintensität dargestellt (vgl. nächste Folie).

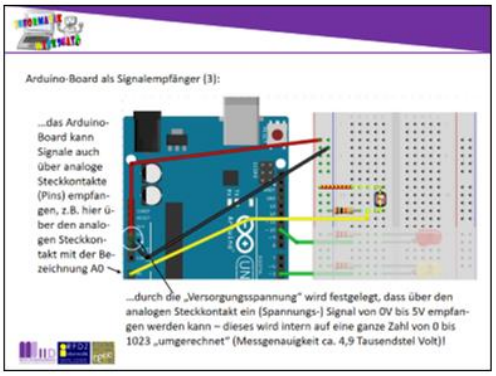


Hinweis: Da in der Informationsdatei bei den Erläuterungen zur Schalterprogrammierung nur die einseitige Verzweigung erläutert wird, sollten diese beiden Folien jedenfalls präsentiert werden!

### Verarbeitung des Photowiderstand-Signals mit einer zweiseitigen Programmverzweigung:

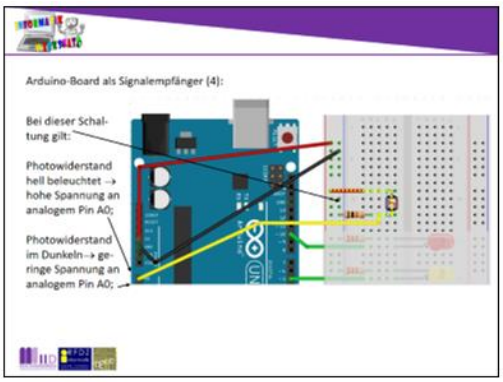
Bei einem Photowiderstand variiert der (Ohm'sche) Widerstand je nachdem, wie hell der Widerstand beleuchtet wird. Da mit Hilfe von Photowiderständen somit der Grad der Helligkeit gemessen werden kann, reicht es nicht, wenn nur zwei unterschiedliche Signale („Spannung" bzw. „keine Spannung") an das Arduino-Board übermittelt werden.

Für die Helligkeitsmessung wird daher ein **analoger Steckkontakt** verwendet, der den anliegenden Spannungspegel in eine ganze Zahl von 0 bis 1023 codiert: Je höher die am Steckkontakt anliegende Spannung ist, desto höher ist die ganze Zahl, die über den Programmcode am Steckkontakt abgerufen werden kann. Details dazu sollten anhand der zugehörigen Inhaltsfolien 6 und 7 (ergänzend zu den Ausführungen in der Informationsdatei) besprochen werden:




Textvorschlag:

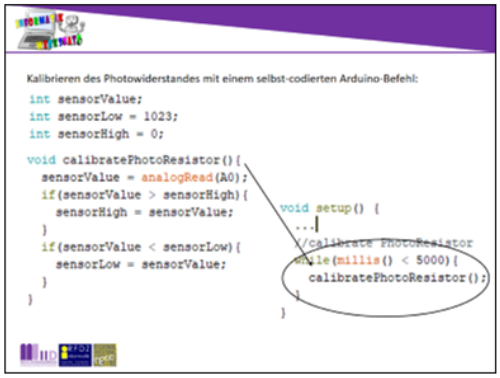
Für den Empfang von Signalen können am Arduino-Board auch analoge Steckkontakte verwendet werden, die nicht nur die Zustände „Spannung liegt an" (**HIGH**) oder „Spannung liegt nicht an" (**LOW**) unterscheiden können, sondern im Bereich der Versorgungsspannung 1024 verschiedene Spannungswerte unterscheiden kann. Der Sensorwert selbst ist dabei immer eine Zahl von 0 bis 1023!




Textvorschlag:

Ein Photowiderstand ist ein elektronischer Bauteil, dessen Widerstand davon abhängt, wie stark er beleuchtet wird. Wenn aber für vorliegende Schaltung im Programm bei einer Verzweigung abgefragt werden soll, ob der Photowiderstand hell beleuchtet ist (oder nicht), d.h. ob am analogen Steckkontakt hohe oder nur geringe Spannung anliegt), muss zuerst der messbare Zahlenbereich von 0 bis 1023 an die herrschenden Helligkeitsverhältnisse angepasst werden – man sagt, der Photowiderstand wird kalibriert (als Weiterleitung zur nächsten Folie)

Neu ist bei der Verwendung eines Bauteils, der **analoge Signale** liefert, dass zuerst festgestellt werden muss, welche Messwerte überhaupt auftreten können – der Bauteil muss zuerst **kalibriert** werden. Der Code des dafür benötigten Befehls sowie deren Verwendung ist sowohl in der Informationsdatei als auch auf den Inhaltsfolien 8 und 9 der Datei „**ST_PR_Arduino_Verzweigungen.pptx**“ erklärt:




Textvorschlag:

Der Code für den Programmbefehl zur Kalibrierung des Photowiderstandes wird vorgegeben (und in einer Arbeitsanregung auch „erforscht“) – für die Bearbeitung der Arbeitsanregungen zur „Programmierung des Photowiderstandes“ ist aber der Aufruf dieses selbst-codierten Befehls im **setup**-Programmteil: In den ersten fünf Sekunden nach Programmstart wird der Befehl **calibratePhotoResistor** in einer Schleife aufgerufen – während dieser Zeit sollte der Photowiderstand (mehrmals) abgedunkelt und dann wieder möglichst hell beleuchtet werden. Dann ist der größte „gemessene“ Sensorwert (im Bereich von 0 bis 1023) in der Variablen **sensorHigh**, der kleinste in der Variablen **sensorLow** gespeichert!


Textvorschlag:

Nach der Kalibrierung des Photowiderstandes kann (kann!) vereinbart werden, dass der Photowiderstand dann „hell beleuchtet“ ist, wenn der am analogen Steckkontakt gemessene Wert größer als der Mittelwert zwischen minimalem und maximalem Messwert ist.
Besonders kann darauf hingewiesen werden, dass bei analogen Steckkontakten der **pinMode** nicht festgelegt werden muss, da diese Steckkontakte bei Aufruf des Befehls **analogRead** automatisch in den Modus **INPUT** versetzt werden.