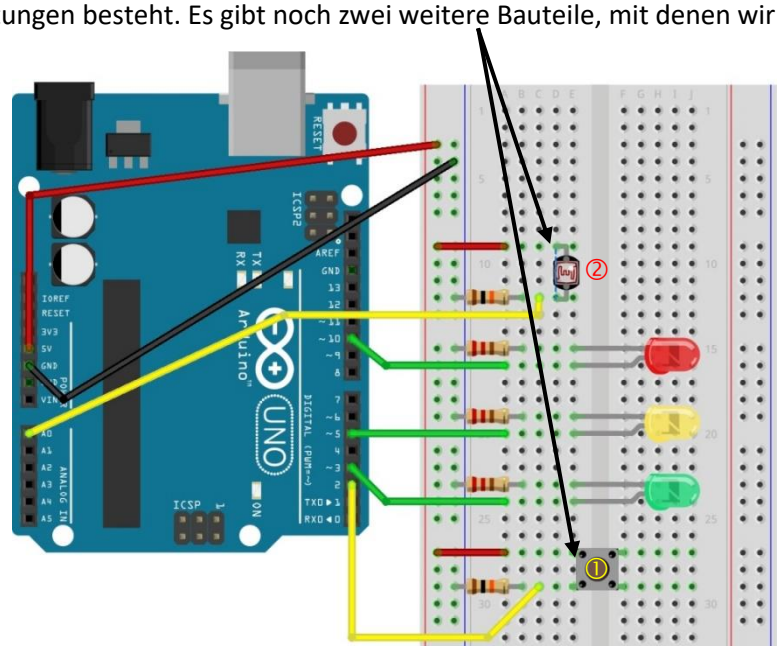
Arduino – ein Programm trifft Entscheidungen

Dir ist sicherlich aufgefallen, dass die Schaltung, die wir verwenden, nicht nur aus den LEDs und den zugehörigen Widerständen bzw. Leitungen besteht. Es gibt noch zwei weitere Bauteile, mit denen wir uns jetzt beschäftigen wollen.

Dabei lernst du, wie ein Arduino-Programm Entscheidungen treffen kann, und dass Signale nicht nur vom Arduino „nach außen“ (OUTPUT) gesendet werden können, sondern dass auch Signalübermittlung „in den Arduino hinein“ (INPUT) möglich ist.



Der mit ① bezeichnete Bauteil ist ein **Schalter**, von dem über die am **digitalen Steckkontakt** mit der Nummer 2 angeschlossene gelbe Signalleitung festgestellt werden kann, ob er gedrückt wurde oder nicht – am **digitalen Steckkontakt** können nämlich genau **zwei Zustände**, z.B. „`HIGH`“ und „`LOW`“, unterschieden werden, die beim Programmieren meist durch 1 (für „`HIGH`“) und 0 (für „`LOW`“) codiert werden.

Bei der vorliegenden Schaltung verhält es sich so:

- Wenn der Druckknopf-Schalter gedrückt wurde, liegt am Steckkontakt mit der Nummer 2 der Spannungspegel „`HIGH`“ an,
- wenn der Druckknopf-Schalter nicht gedrückt wurde, liegt am Steckkontakt mit der Nummer 2 der Spannungspegel „`LOW`“ an.

Genau dieser Sachverhalt muss durch ein Programm, mit dem auf das Drücken des Schalters „reagiert“ werden soll, abgebildet werden:

Dazu wird eine Variable **`buttonPin`** vereinbart, in der die Nummer des Kontakts gespeichert, über den beim Drücken des Druckknopfes das Signal an das Arduino-Board gesendet wird. In unserer Schaltung ist das der Kontakt Nummer 2, daher lautet der Befehl **`int buttonPin = 2;`**

```
...
int buttonPin = 2;
int switchstate = 0; //!
...
```

Der „Zustand“ des Druckknopfs wird mit der (Ganzzahl-) Variablen **`switchstate`** „gemerkt“. Wurde der Druckknopf nicht gedrückt, hat diese Variable den Wert 0 (oder „`LOW`“), andernfalls den Wert 1 (oder „`HIGH`“). Zu Beginn hat diese Variable den Wert 0.

Dass über den Kontakt mit der Nummer 2 Signale an den Arduino gesendet werden können, wird im `setup()`-Teil mit dem Befehl `pinMode(buttonPin, INPUT);` vereinbart.

```
void setup() {
  pinMode(redLedPin,OUTPUT);
  pinMode(yellowLedPin,OUTPUT);
  pinMode(greenLedPin,OUTPUT);
  pinMode(buttonPin,INPUT); //!
  //set inital LED-states
  digitalWrite(redLedPin,HIGH);
  digitalWrite(yellowLedPin,LOW);
  digitalWrite(greenLedPin,LOW);
}
void loop() {
  switchstate = digitalRead(2); //!
  if(switchstate == HIGH){      //!
  }
}
```

Der jeweils aktuelle „Zustand“ des Schalters wird der Variablen **`switchstate`** im `loop()`-Teil mit dem Befehl

```
switchstate = digitalRead(2);
```

zugewiesen.

(`switchstate = digitalRead(buttonPin);` wäre ebenso richtig).

Ob (engl.: „**`if`**“) der Druckknopf tatsächlich gedrückt wurde, wird schließlich in der Programmzeile entschieden, die mit dem Befehl `if` (dt.: „ob“) beginnt…

…Code(block), der ausgeführt wird, wenn in der Variablen **`switchstate`** der Wert `HIGH` gespeichert ist – das ist genau dann der Fall, wenn der Schalter gedrückt wurde (und so ein Signal an den Steckkontakt mit Nummer 2 gesendet hat)…

Diesen Befehl liest du am besten so:

> Überprüfe, ob die in Klammer nach dem Schlüsselwort `if` stehende Bedingung erfüllt ist.
>
> Nur wenn dies der Fall ist, werden die im **Programmblock** nach **`if (Bedingung)`** codierten Befehle ausgeführt.

Wie gewohnt ist dieser **„Bedingung-ist-erfüllt“-Programmblock** durch ein Paar geschwungener Klammern, **{ }**, begrenzt.

Die Bedingung

```
switchstate == HIGH
```

ist nur dann erfüllt, wenn in der Variablen **`switchstate`** der Wert 1 („`HIGH`“) „gemerkt ist“, d.h., wenn zuvor der Druckknopfschalter gedrückt wurde.
Man sagt, in diesem Fall liefert die Bedingung **`switchstate == HIGH`** den (Wahrheits-) Wert `true`, und nennt den daraufhin ausgeführten Programmblock auch den „`true`“-**Programmblock**.

WICHTIG: Beachte das doppelte Gleichheitszeichen **==** beim Vergleich in der Klammer und unterscheide es von der Zuweisung eines Wertes an eine Variable.

…mit diesen Informationen kannst du dich bereits den **Arbeitsanregungen zur Schalterprogrammierung im Arbeitsblatt** widmen!

Bearbeite diese Arbeitsanregungen, bevor du

☞ **hier weiterliest,** um zu erfahren, wie du den mit ② bezeichnete Bauteil, einen **Photowiderstand,** programmieren kannst.

Ein **Photowiderstand** heißt deswegen so, weil sein Widerstandswert davon abhängt, wie hell er beleuchtet wird. Der Photowiderstand in unserer Schaltung ist über eine gelbe Signalleitung mit dem **analogen Steckkontakt** A0 verbunden. Über diesem Steckkontakt kann die Spannung ausgelesen werden, die „hinter" dem Photowiderstand „übrigbleibt". Es gilt

Photowiderstand beleuchtet → geringer Widerstandswert → hohe Spannung an A0
Photowiderstand im Dunkeln → hoher Widerstandswert → geringe Spannung an A0

Dabei ist noch ein wesentlicher **Unterschied** zwischen **digitalen Steckkontakten**, die auf INPUT geschaltet sind, und **analogen Steckkontakten** zu beachten:

Während am **digitalen Steckkontakt** nur zwischen „Spannung liegt an" (Wert „`HIGH`", entsprechend 1, oder „keine Spannung liegt an" (Wert „`LOW`", entsprechend 0),

kann der Wert der Spannung, der über einen **analogen Steckkontakt** gemessen wird, von 0 V bis 5 V variieren. Eine Besonderheit analoger Steckkontakte ist zudem, dass die an ihnen anliegende Spannung in eine Zahl von 0 bis 1023 umgerechnet wird. Dadurch wird der Eindruck vermittelt, dass ein kontinuierliche (analoge) Signale mit „**beliebig vielen**" **Zwischenwerten** gemessen werden können. „**Beliebig viele Zwischenwerte**" bedeutet im vorliegenden Fall, dass die am **analogen Steckkontakt** A0 anliegende Spannung mit einer Genauigkeit von etwa 5 V/1024 ≈ 0,00488 V, also 4,9 Millivolt, gemessen werden kann.

Die grundlegende **Programmidee** bei der Programmierung der „Photowiderstandschaltung" ist: Wenn der Photowiderstand (hell) beleuchtet wird, soll etwas anderes „passieren", als wenn der Photowiderstand (ziemlich) im Dunkeln ist…

Mit dem bereits bekannten „Entscheidungs-Befehl" (und einer kleinen Erweiterung) lässt sich das in „**Pseudocode**" so formulieren:

```
if(Wert an Pin A0 ist hoch) {
      // Block der Befehle, die auszuführen sind, wenn
      // der Photowiderstand hell beleuchtet ist
}
else {
      // Block der Befehle, die auszuführen sind, wenn
      // der Photowiderstand im Dunkeln ist
}
```

Die „kleine Erweiterung“ ist das Befehlswort **`else`** (engl.: „andernfalls“), das jenen Befehlsblock einleitet, der ausgeführt werden soll, wenn die bei **`if`** überprüfte **Bedingung nicht erfüllt** ist (d.h. den Wahrheitswert **`false`** liefert).

Es bleibt noch zu klären, wie „hell beleuchtet“ zu verstehen ist – eine starke Leselampe gibt sicher helleres Licht als eine gewöhnliche Raumbeleuchtung, die aber wohl ebenfalls nicht als „dunkel“ empfunden wird.

Eine Möglichkeit, dies zu klären, besteht darin, zuallererst zu messen, welche Spannung am analogen Steckkontakt A0 anliegt, wenn die Beleuchtung in der jeweiligen Situation größtmöglich bzw. kleinstmöglich (z.B. wenn der Photowiderstand abgedeckt wird) ist.

```
int redLedPin = 10;
int yellowLedPin = 5;
int greenLedPin = 3;
int sensorValue;
int sensorLow = 1023;
int sensorHigh = 0;

void calibratePhotoResistor(){
  sensorValue = analogRead(A0);
  if(sensorValue > sensorHigh){
    sensorHigh = sensorValue;
  }
  if(sensorValue < sensorLow){
    sensorLow = sensorValue;
  }
}
```

Dies wird durch einen selbst programmierten Befehl namens **`calibratePhotoResistor()`** erledigt, dessen Unterprogrammcode nebenstehend abgebildet ist, und das dafür sorgt, dass der kleinstmögliche Spannungswert in der Variablen namens **`sensorLow`** und der größtmögliche Spannungswert in der Variablen **`sensorHigh`** gespeichert wird.

Dieser Befehl ist sinnvollerweise im **`setup()`**-Block des Arduinoprogramms aufzurufen – in nebenstehendem Codefragment wird er in einer Schleife, die fünf Sekunden lang durchlaufen wird, immer wieder aufgerufen und misst dabei die am **analogen Steckkontakt A0** anliegende Spannung – der dabei verwendete vordefinierte Befehl **`millis()`** zählt dabei, wie viele Millisekunden (also tausendstel Sekunden) seit dem Beginn des Schleifendurchlaufs vergangen sind.

```
void setup() {
  ...
  //calibrate PhotoResistor
  while(millis() < 5000){
    calibratePhotoResistor();
  }
}
```

Während dieser fünf Sekunden sollte der Photowiderstand abwechselnd hell beleuchtet bzw. abgedunkelt werden, damit in den Variablen **`sensorLow`** und **`sensorHigh`** relevante (kleinst- bzw. größtmögliche) Spannungswerte gespeichert werden, die den herrschenden Beleuchtungsverhältnissen entsprechen. Mit diesen Werten kann dann vereinbart werden, dass der Photowiderstand „hell beleuchtet“ ist, wenn z.B. am **analogen Steckkontakt A0** ein Spannungswert anliegt, der größer als **`(sensorHigh + sensorLow)/2`** ist.

Ein entsprechendes Codefragment für den loop()-Block des Arduino-Programms ist nebenstehend abgebildet.

```
void loop() {
  sensorValue = analogRead(A0);
  ...
  if(sensorValue > (sensorHigh + sensorLow)/2){
   ...
  }
  else{
    ...
  }
}
```

…mit diesen Informationen solltest du nun auch die **Arbeitsanregungen zur Programmierung des Photowiderstands im Arbeitsblatt** erfolgreich erledigen können 😊.