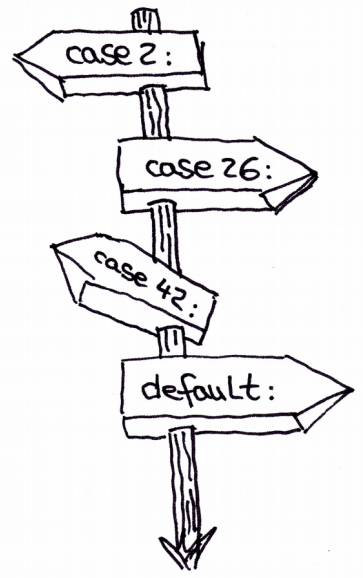
INFORMATIK

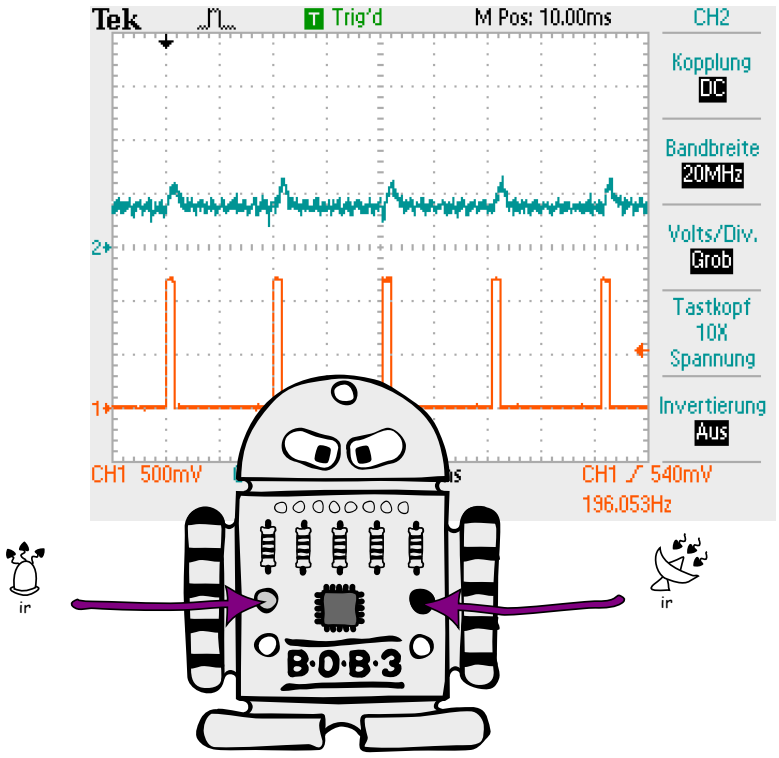
Programmieren lernen mit BOB3

Lernbegleitheft + Arbeitsblätter - Sekundarstufe I
Einführung in die textuelle Programmierung mit dem Roboter BOB3

Modul 3

Funktionen & Sensorik





Modul 3

Funktionen & Sensorik

Dieses Dokument dient zur Übersicht des Kurses „Programmieren lernen mit BOB3“ für die Sekundarstufe I. Es werden die Lehr- und Lernmaterialien, die benötigten Zeiteinheiten und Vorschläge für konkrete Unterrichtseinheiten gegeben. Im Folgenden ist eine Unterrichtseinheit für 45 min ausgelegt.

Übersicht Modul 3:

Modul 3 umfasst insgesamt 12 Unterrichtseinheiten mit je ca. 45 Minuten

Lehrerbegleitheft mit Konzepten und Ablaufplänen der einzelnen Einheiten

25 Programmier-Mikrolerneinheiten:

Die Schüler vertiefen das Prinzip und die Anwendungsmöglichkeiten von Variablen und setzen das erlernte Wissen in kleinen Programmbeispielen um. In verschiedenen Experimenten mit den Multifeld-Touch-Sensoren verwenden sie Variablen, um die jeweils aktuellen Sensorwerte abzuspeichern und während des Programmablaufs zur Verfügung zu stellen. In diesem Zusammenhang beschäftigen sie sich intensiver mit der genauen Funktionalität der Sensoren und setzen sich mit dem Zeit-Multiplex-Verfahren auseinander. Als weitere Kontrollstrukturen lernen die Schüler ‚Switch-Case-Verzweigungen‘ und ‚While-Schleifen‘ kennen und vertiefen ihr Wissen zur Kontrollstruktur ‚if / else‘ anhand einer Leistungsüberprüfung. Die Schüler erlernen den Sinn und Zweck von Funktionen und erarbeiten den Unterschied von Funktionen mit und ohne Parameter. In verschiedenen Experimenten mit dem IR-Sensor werden Funktionen mit Rückgabewert konkret zur Auswertung eingesetzt. In diesem Zusammenhang erwerben die Schüler weiteres Wissen zur Funktionalität des IR-Sensors, insbesondere zum Reflexionsverfahren. Anschließend definieren die Schüler eigene Funktionen und diskutieren das aus diesen Möglichkeiten resultierende Potential. Zum Abschluss der Lerneinheit werden offene Aufgaben, wie z.B. die Programmierung eines Parksensors angeboten.

- Arbeitsblatt 9 - „***Variablen***“ + Lösungen zum AB 9
- Arbeitsblatt 10 - „***Touch-Sensoren***“ + Lösungen zum AB 10
- Arbeitsblatt 11 - „***Switch-Case***“ + Lösungen zum AB 11
- Arbeitsblatt 12 - „***Funktionen***“ + Lösungen zum AB 12
- Arbeitsblatt 13 - „***While-Schleife***“ + Lösungen zum AB 13
- Arbeitsblatt 14 - „***IR-Sensor***“ + Lösungen zum AB 14

www.bob3.org

OER-Materialien: http://www.bob3.org/mint
Hauptseite der Lerneinheiten: http://www.progBob.org

1. Unterrichtseinheit

In der ersten Unterrichtseinheit vertiefen die Schülerinnen und Schüler ihr Wissen zur Theorie und Anwendung von **Variablen**. Sie arbeiten mit den Multifeld-Touch-Sensoren der Arme von BOB3, **deklarieren** sich in diesem Zusammenhang Variablen vom **Datentyp Integer** und lernen, was die **Zuweisung** eines neuen Wertes bedeutet.

Zeitbedarf: ca. 45 min

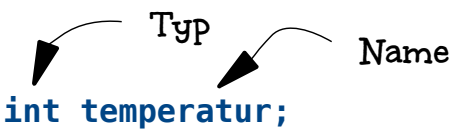
Vorkenntnisse: BOB3 Modul 1+2

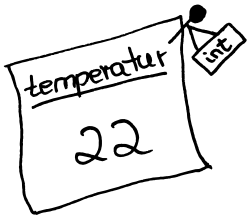
Material: Papier, Stift, Arbeitsblatt 9
PC/Laptop/Tablet, BOB3 mit Helm/Dock

Ablauf

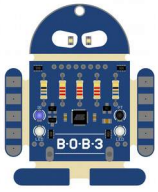
Die Schüler bearbeiten das **Arbeitsblatt 9** und lernen die Eigenschaften, das Konzept und die Anwendungsmöglichkeiten von **Variablen** kennen:

„Eine Variable ist ein Speicherort für Zahlen, Zeichen oder sonstige Daten“



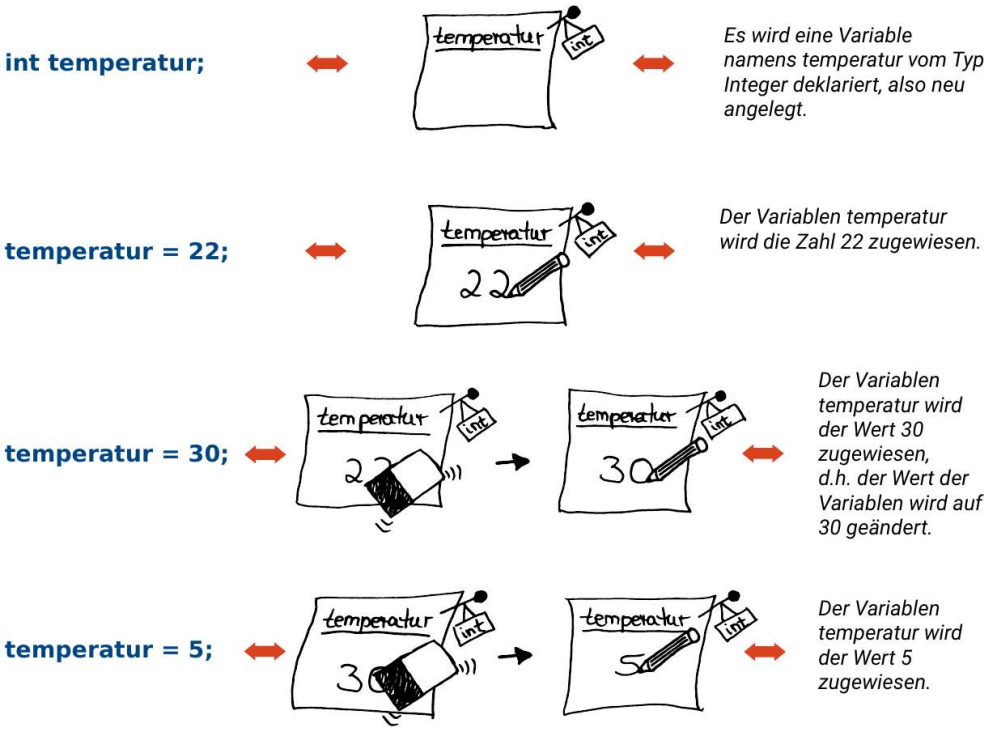


Die SuS lernen, wie **Variablen** definiert werden und welche Eigenschaften (**Name**, **Datentyp**) sie haben. Die Schüler arbeiten zunächst nur mit dem Datentyp **Integer** und lernen, dass dieser für ganze Zahlen verwendet wird. Um eine Variable zu **deklarieren**, also neu einzuführen, schreibt man für eine Integer Variable das Schlüsselwort `int` gefolgt von dem Variablen-Namen und einem Semikolon. Nach der Deklaration einer Variablen wird ihr Name und ihr Datentyp nicht mehr verändert, der jeweils gespeicherte Wert ist jedoch veränderbar!



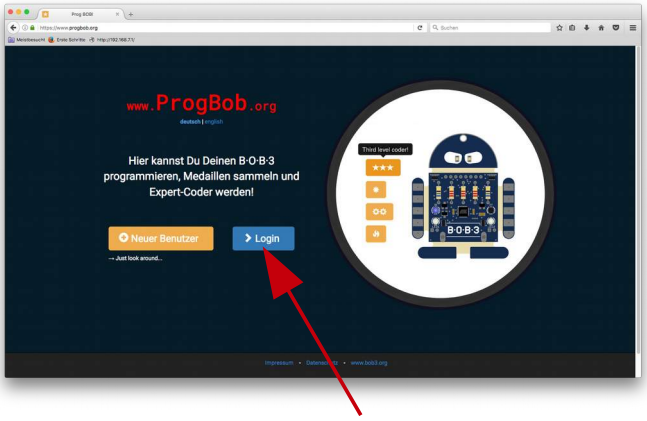
www.bob3.org

Im Gegensatz zu **Konstanten** sind **Variablen** variabel! Die Schüler lernen, dass der Wert einer Variablen beliebig oft geändert werden kann. Dieses Konzept und die Funktionsweise von Variablen kann durch folgendes **Tafelbild** veranschaulicht und zusammen mit den Schülern erarbeitet werden:



Anschließend bearbeiten die SuS die Aufgaben 1 – 6 des Arbeitsblatts und besprechen ihre Lösungen.

Die Schüler loggen sich mit ihren jeweiligen Accounts auf der Seite **http://www.ProgBob.org** ein und starten das **„Intro III“**-Kapitel:



!! Wichtig: erneutes Einloggen mit bereits vorhandenen Daten über „**Login**“ !!

www.bob3.org

Die Schüler bearbeiten die **erste** und die **zweite Lerneinheit** und vertiefen mit einem kurzen Beispiel zur Ansteuerung der **Arm-Sensoren** das zuvor erworbene Wissen:

```
1  #include <BOB3.h>
2
3  void setup() {
4
5  }
6
7  void loop() {
8
9    // Status der Arme abfragen:
10   int wert1 = bob3.getArm(1);
11   int wert2 = bob3.getArm(2);
12
13   if (wert1 != 0) {
14     bob3.setLed(LED_3, ON);
15   } else {
16     bob3.setLed(LED_3, OFF);
17   }
18
19   // schreibe hier den fehlenden Quellcode
20
21
22
23
24
25
26
27 }
28
```

In der Lerneinheit wird zunächst ein Programmteil zur **Abfrage von Arm 1** vorgegeben, der von den Schülern ausprobiert wird. Sobald Arm 1 berührt wird, soll LED3 eingeschaltet werden. Hierfür wird eine **Integer Variable *wert1*** deklariert, die den jeweils aktuellen Wert des Sensors Arm 1 zugewiesen bekommt. In Zeile 13 des Programms wird die Variable innerhalb einer if-Abfrage **verwendet**, um zu entscheiden, ob LED3 eingeschaltet wird oder nicht.

In der **Aufgaben-Einheit** des zweiten Teils bekommen die SuS die Arbeitsanweisung, das Programm so zu ergänzen, dass eine **weitere Integer Variable** die Sensorwerte von Arm2 speichert. Um anzuzeigen, ob Arm2 berührt wird oder nicht, programmieren die Schüler ab Zeile 20 selbständig den fehlenden Quellcode und verwenden hierbei ihre neu eingeführte Variable.

Arm 2

___info___

Jetzt soll BOB3 auch noch auf Berührungen am **Arm 2** reagieren!

Falls **Arm 2 berührt** wird, dann soll die weiße **Bauch-LED 4** leuchten.

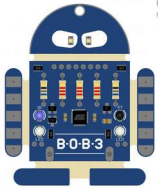
In Zeile 11 fragen wir den **Status von Arm 2** ab und speichern den **Rückgabewert** in der Variablen `wert2`.

Arm 2

___aufgabe___

Ergänze den Quellcode ab Zeile 20, so dass die LED 4 von Bob beim Berühren von Arm 2 leuchtet.

Compiliere dein Programm und teste es auf dem BOB3.



www.bob3.org

2. + 3. Unterrichtseinheit

In der zweiten und dritten Unterrichtseinheit bearbeiten die Schülerinnen und Schüler die **Lerneinheiten drei**, **vier** und **fünf** des „**Intro III**“-Kapitels, die sich ebenfalls mit den **Multifeld-Touch-Sensoren** und **Variablen** beschäftigen. Sie experimentieren mit verschiedenen Programmen und vertiefen insbesondere ihre Kenntnisse über **Rückgabewerte** von **Methoden**. Als Anwendung des Gelernten bietet die fünfte Lerneinheit das Programm „**Intelligente Taschenlampe**“, bei dem die Schüler ihr Wissen über **Variablen** und ‚**if-else**‘-Strukturen vertiefen.

Zeitbedarf: ca. 90 min

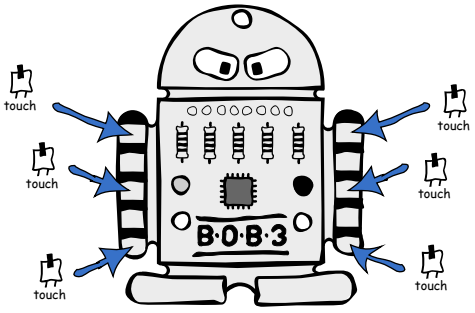
Vorkenntnisse: BOB3 Modul 1+2

Material: Papier, Stift, Arbeitsblatt 10
PC/Laptop/Tablet, BOB3 mit Helm/Dock

Ablauf

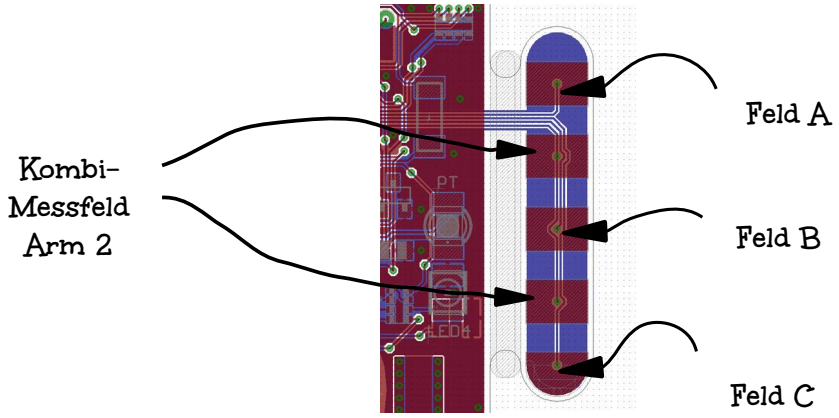
Die Schüler bearbeiten das **Arbeitsblatt 10** und lernen die technischen Hintergründe der Multifeld-Touch-Sensoren und deren Ansteuerung über die Methoden der Software-Bibliothek kennen und vertiefen:

„Bob‘s Arme sind Multifeld-Touch-Sensoren, die aus Mess- und Aktivierungsfeldern bestehen und mit dem Zeit-Multiplex-Verfahren angesteuert werden.“



Die SuS lernen, aus welchen Bestandteilen die Sensoren aufgebaut sind und wie diese angesteuert werden. Das Zeit-Multiplex-Verfahren ist eine

Methode zur Signalübertragung, die verwendet wird, um viele Signale an wenigen Eingängen des Mikrocontrollers anzuschließen. Dabei teilen sich mehrere Signale einen Eingang: Der Controller von BOB3 hat zwei Eingänge, einen für Arm 1 und einen für Arm 2, die Arme können jedoch an insgesamt sechs verschiedenen Stellen angefasst werden und liefern also sechs Signale!
Jeder Arm besteht jeweils aus fünf Feldern: 3 **Aktivierungsfelder** (A, B, C) und 2 **Messfelder**. Sobald ein Aktivierungsfeld gleichzeitig mit einem Messfeld berührt wird, bekommt der Bob ein Signal, ob Feld A, Feld B oder Feld C berührt wurde.



Die Schüler bearbeiten die Aufgaben 1 – 5 des Arbeitsblatts und besprechen ihre Lösungen.

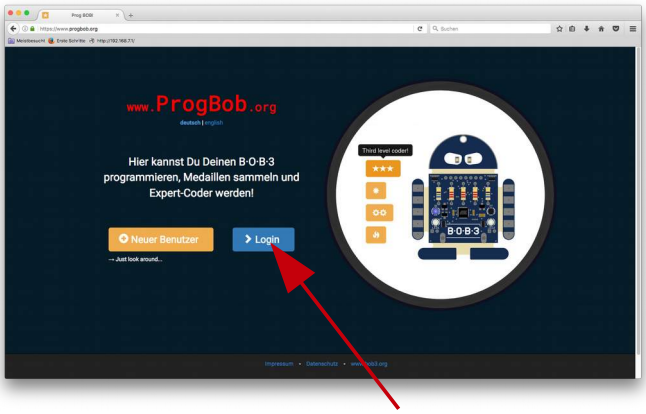
Zur **Ansteuerung** der Armsensoren stehen in der Software-Bibliothek des BOB3 fertig implementierte Methoden zur Verfügung. Die Methode `bob3.getArm(id)` liefert den **aktuellen Wert** des jeweiligen Sensors und hat somit vier verschiedene mögliche **Rückgabewerte** und zwei mögliche **Parameter**:

Rückgabewert	0	1	2	3
Bedeutung	Keine Berührung	Berührung oben	Berührung mittig	Berührung unten

id	1	2
Sensor	Arm 1	Arm 2

Falls die SuS im Anschluss an die Lerneinheiten von Intro II das Kapitel ‚**Freundschaftstester**' bearbeitet hatten, kann an dieser Stelle noch thematisiert werden, dass mit der Methode `bob3.enableArms(ARMS_DETECTOR)` die Sensoren z.B. für den verwendeten **Friend-Detection-Mode** aktiviert bzw. deaktiviert werden. Dies ist ein anderes Verfahren, bei dem die Leitfähigkeit der Finger gemessen wird: Alle Aktivierungsfelder werden hierbei gleichzeitig aktiviert und an den beiden Messfeldern wird der Stromfluss gemessen.

Die SuS starten den Webbrowser, gehen auf die Seite **http://www.ProgBob.org**, loggen sich mit Ihrem Account ein und gehen zum **„Intro-III“**-Kapitel:



„Login“

In der **dritten** und **vierten** Lerneinheit soll die genaue **Position** der Berührung detektiert werden. Je nachdem, ob der Arm *oben*, *mittig* oder *unten* berührt wird, soll Bob unterschiedlich reagieren. Die Schüler arbeiten mit der Methode **`bob3.getArm()`** und können mit verschiedenen Rückgabewerten und Parametern experimentieren.

```
#include <BOB3.h>

void setup() {

}

void loop() {
  int wert1 = bob3.getArm(1);

  // Arm1 oben:
  if ( wert1 == 1 ) {
    bob3.setEyes(WHITE, WHITE);
    delay(200);
    bob3.setEyes(OFF, OFF);
    delay(200);
  }


  // Arm1 unten:
  if ( wert1 == 3 ) {
    bob3.setWhiteLeds(ON, ON);
    delay(200);
    bob3.setWhiteLeds(OFF, OFF);
    delay(200);
  }


}
```

info

Jetzt wollen wir die **genaue Position** der Berührung detektieren, dafür müssen wir die Methode `bob3.getArm()` näher kennenlernen:

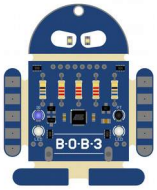
Wie du schon weisst, schreiben wir in die **runden Klammern** die Nummer des Arms, den wir abfragen wollen.

Wir schreiben `bob3.getArm(1)`, um den **Arm 1** Sensor von BOB3 abzufragen.

info

Wichtig zu wissen: Die Methode `bob3.getArm()` hat vier verschiedene **Rückgabewerte**:

- `0` : der Arm wird **nicht** berührt
- `1` : der Arm wird **oben** berührt
- `2` : der Arm wird **mittig** berührt
- `3` : der Arm wird **unten** berührt



www.bob3.org

Als Beispiel aus der **eigenen Erlebniswelt** sollen die Schüler nun mit dem erlernten Wissen eine ‚**Intelligente Taschenlampe**' programmieren. Dabei soll Arm 1 wie ein **Schieberegler** bedient werden: Wird der Arm nur oben berührt, dann sollen erstmal nur beide Augen in weiss eingeschaltet werden. Bei mittiger Berührung geht eine weitere LED weiss an, die volle Helligkeitsstufe wird erreicht, wenn der Sensor unten berührt wird, dann sind alle Lampen an!

```
#include <BOB3.h>

void setup() {

}

void loop() {

  // Arm 1 abfragen:
  int wert1 = bob3.getArm(1);

  if (      ) {


  }


  else if (      ) {


  }


  else if (      ) {


  }


  else {


  }

}
```

Arm 5

___info___

Jetzt programmieren wir Bob als **intelligente Taschenlampe**!

Wir verwenden den Arm 1 wie einen **Schieberegler**:

Wenn man Arm 1 **oben** berührt, gehen erstmal nur **beide Augen** in weiss an. Damit die Taschenlampe **heller** leuchtet, berührt man den Arm 1 **mittig**, dann geht **zusätzlich** noch die **Bauch-LED 3** an. Die **volle Helligkeitsstufe** wird erreicht, indem man Arm 1 unten berührt, dann hat Bob **alle vier LEDs weiss an**!

___aufgabe___

Programmiere die **schlaue Taschenlampe**!

Füge die folgenden Teile an den richtigen Stellen ein:

```
wert1 == 1
```

```
bob3.setEyes(WHITE, WHITE);
```

```
wert1 == 2
```

```
bob3.setEyes(WHITE, WHITE);
```

```
bob3.setWhiteLeds(ON, OFF);
```

```
wert1 == 3
```

```
bob3.setEyes(WHITE, WHITE);
```

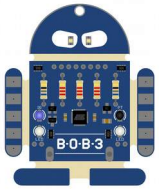
```
bob3.setWhiteLeds(ON, ON);
```

```
bob3.setEyes(OFF, OFF);
```

```
bob3.setWhiteLeds(OFF, OFF);
```

Compiliere dein Programm und teste es mit BOB3!

Die SuS implementieren das Programm unter Verwendung einer ‚if-else if-else'-Struktur, einer Integer Variablen zur Speicherung des Sensorwertes und der Methode **`bob3.getArm(id)`**. Sie vertiefen hierbei den Sinn und die Anwendung von **Variablen** und den Umgang mit **Parametern** und **Rückgabewerten** von Methoden. Durch die begreifbare und anwendungsorientierte Aufgabenstellung und das direkt überprüfbare Feedback an der konkreten Hardware stellt sich ein motivierender Lernerfolg ein.



www.bob3.org

4. + 5. Unterrichtseinheit

In der vierten und fünften Unterrichtseinheit bearbeiten die Schülerinnen und Schüler eine **Leistungsüberprüfung** (Lerneinheit 6) und die **Lerneinheiten sieben** und **acht** des **„Intro III“**-Kapitels, die sich mit der Programmierstruktur ‚**Switch-Case**‘ beschäftigen. Sie lernen, in welchen Fällen eine Switch-Case Verzweigung einer ‚if-else‘ Verzweigung vorzuziehen ist und wie diese implementiert wird. In der Leistungsüberprüfung programmieren die Schüler Bob als **Obstsortiermaschine**, wobei sie selbständig eine **Variable** deklarieren und ihr den **Rückgabewert** aus einem Methodenaufruf zuweisen. Vervollständigt wird das Programm durch eine ‚**if-else**‘ Struktur für die Sortierung.

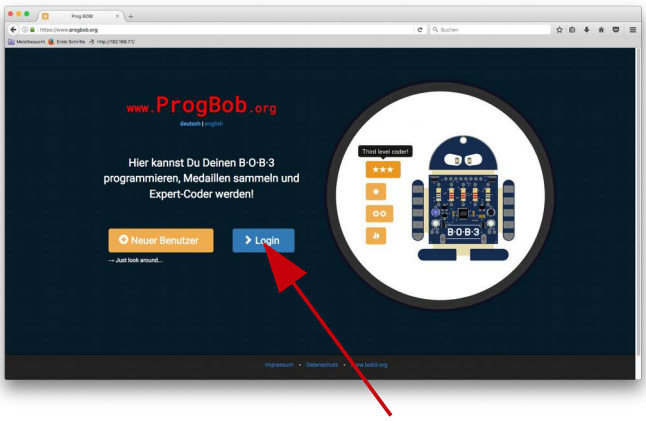
Zeitbedarf: ca. 90 min

Vorkenntnisse: BOB3 Modul 1+2

Material: Papier, Stift, Arbeitsblatt 11
PC/Laptop/Tablet, BOB3 mit Helm/Dock

Ablauf

Die SuS starten den Webbrowser, gehen auf die Seite **http://www.ProgBob.org**, loggen sich mit Ihrem Account ein und gehen zum **„Intro-III“**-Kapitel:



„Login“

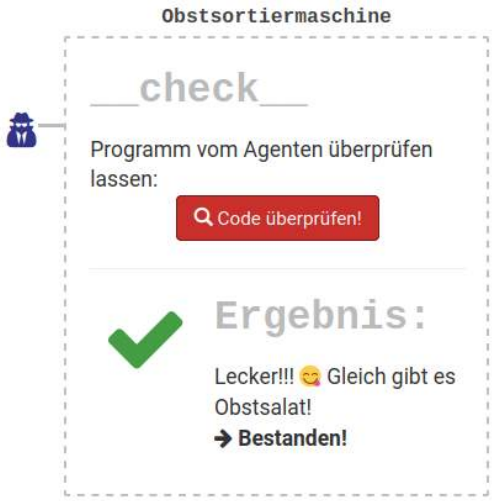
Leistungsüberprüfung

Die Schüler wählen Teil sechs des Intro-III Kapitels, lesen die Aufgabenstellung zur Programmierung der **Obstsortier-maschine** und erarbeiten eine Lösung.

Lernziele:

1) *Aufgabenstellung algorithmisch umsetzen*
2) *Integer Variable deklarieren*
3) *Armsensor mit passender Methode abfragen*
4) *Rückgabewert in Variable speichern*
5) *If-else if- else Struktur anwenden*
6) *Vergleichsoperatoren anwenden*
7) *LEDs farbig ansteuern*


```
#include <BOB3.h>


void loop() {

  // Arm 1 abfragen:
  int wert1 = bob3.getArm(1);


  // Kirschen
  if ( wert1 == 1 ) {
    bob3.setEyes(RED, RED);
  }

  // Orangen
  else if ( wert1 == 2 ) {
    bob3.setEyes(ORANGE, ORANGE);
  }

  // Kiwi
  else if ( wert1 == 3 ) {
    bob3.setEyes(GREEN, GREEN);
  }

  // kein Obst
  else {
    bob3.setEyes(OFF, OFF);
  }


}

```



Die Schülerlösung wird automatisch geprüft und zählt als bestanden, sobald die Aufgabenstellung korrekt gelöst wurde und das Programm compiliert.

www.bob3.org

Die Schüler bearbeiten das **Arbeitsblatt 11** und lernen die Kontrollstruktur ‚**switch-case**' als **Verzweigung** kennen, die dazu dient, viele verschiedene Fälle zu unterscheiden:

„Eine switch-case-Struktur ist eine Verzweigung"

```
switch (Variable){
  case WERT1:
  Anweisungen1;
  break;

  case WERT2:
  Anweisungen2;
  break;

  case WERT3:
  Anweisungen3;
  break;

  case WERT4:
  Anweisungen4;
  break;

  case WERT5:
  Anweisungen5;
  break;

  case WERT6:
  Anweisungen6;
  break;

  default:
  Anweisungen7;
  break;
}
```


```
3  void loop() {
4
5    int anzahlLampen = 3;
6
7    switch (anzahlLampen) {
8      case 1:
9        bob3.setEyes(WHITE, OFF);
10       break;
11
12     case 2:
13       bob3.setEyes(WHITE, WHITE);
14       break;
15
16     case 3:
17       bob3.setEyes(WHITE, WHITE);
18       bob3.setWhiteLeds(ON, OFF);
19       break;
20
21     case 4:
22       bob3.setEyes(WHITE, WHITE);
23       bob3.setWhiteLeds(ON, ON);
24       break;
25
26     default:
27       bob3.setEyes(OFF, OFF);
28       bob3.setWhiteLeds(OFF, OFF);
29       break;
30   }
31
32 }
```

Die Schüler bearbeiten die Aufgaben 1 – 7 des Arbeitsblatts und besprechen ihre Lösungen.

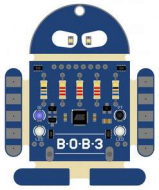
Die Kontrollstruktur wird mit dem Schlüsselwort **switch** eingeleitet, es folgen die verschiedenen Fälle (**case**) und die jeweils auszuführenden Anweisungen. Die Schüler lernen die Verwendung von **break-Anweisungen** und des optionalen **default**-Zweigs.

Aufgabe 1 und Aufgabe 2 wiederholen die Theorie, Aufgabe 3 und Aufgabe 4 intensivieren das Gelernte mit Quellcode-Fragen und die Aufgabe 5 liefert ein konkretes **Anwendungs-Beispiel**: Die Schüler sollen zunächst herausfinden, was das Programm bewirkt und beantworten dazu passende Fragen.

Beschreibe ausführlich die Funktion des folgenden Programms:

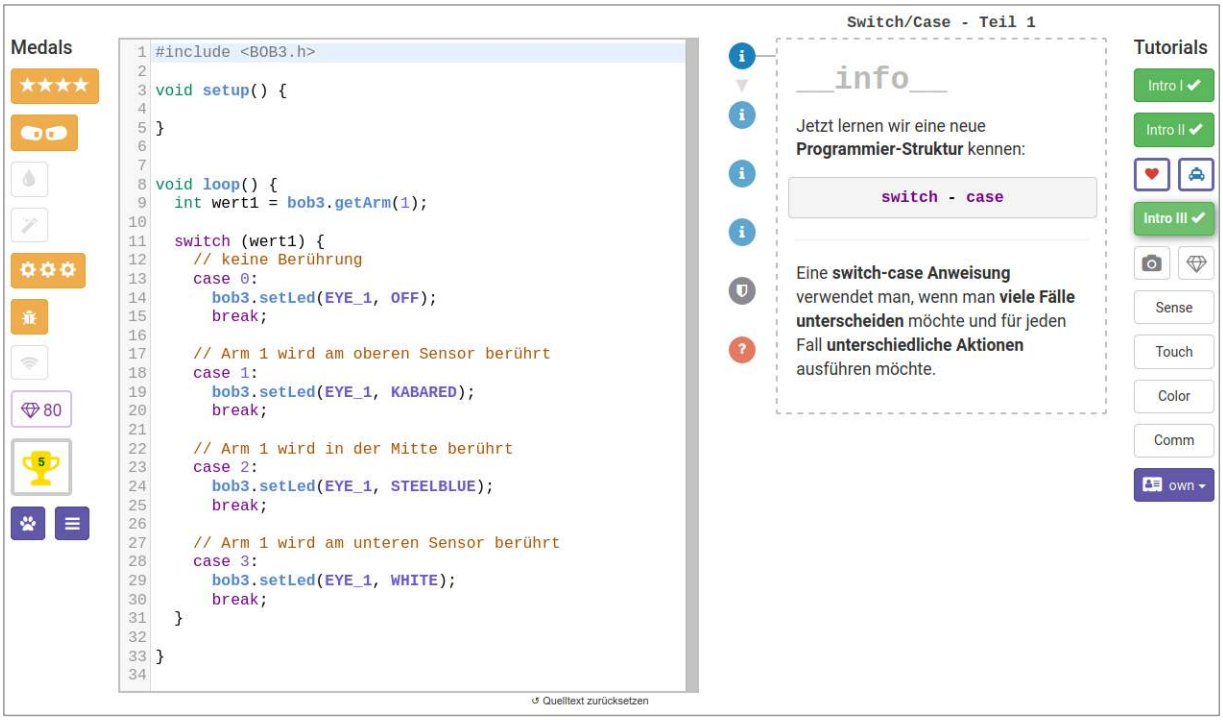
```
8  void loop() {
9    int wert1 = bob3.getArm(1);
10
11   switch (wert1) {
12     case 0:
13       bob3.setLed(EYE_1, OFF);
14       bob3.setLed(EYE_2, OFF);
15       break;
16
17     case 1:
18       bob3.setLed(EYE_1, RED);
19       bob3.setLed(EYE_2, RED);
20       break;
21
22     case 2:
23       bob3.setLed(EYE_1, ORANGE);
24       bob3.setLed(EYE_2, ORANGE);
25       break;
26
27     case 3:
28       bob3.setLed(EYE_1, GREEN);
29       bob3.setLed(EYE_2, GREEN);
30       break;
31   }
32
33   delay(50);
34 }
35
```

BOB3 arbeitet als Ampelmännchen:
~~Wenn Arm 1 oben gedrückt wird, dann leuchten beide Augen~~ rot, wird der Arm in der Mitte gedrückt, leuchten beide Augen orange, wird der Arm unten gedrückt, leuchten beide Augen ~~grün. Ohne Berührung sind die Augen aus!~~



www.bob3.org

Anschließend bearbeiten die Schüler die Lerneinheiten **sieben** und **acht** des **Intro-III Kapitels**. Im ersten Teil wenden sie ihr erworbenes Wissen zur neuen Programmierstruktur an. Das Quellcode-Beispiel verwendet eine Integer Variable **`wert1`**, um den Rückgabewert der Methode **`bob3.getArm(1)`** zur Abfrage von Arm 1 zu speichern. In der folgenden ‚**switch-case**'-Verzweigung wird pro möglichem Rückgabewert ein Fall (**case**) erzeugt, so dass je nach Rückgabewert verschiedene **Aktionen** ausgeführt werden. Je nachdem wo Arm 1 berührt wird, wird das Auge 1 in einer anderen Farbe eingeschaltet. Die Schüler lernen, dass jeder Zweig mit einer **break-Anweisung** abgeschlossen wird und was die Anweisung zur Folge hat!

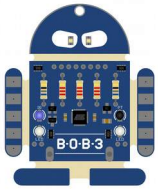



```ino
#include <BOB3.h>

void setup() {

}


void loop() {
  int wert1 = bob3.getArm(1);

  switch (wert1) {
    // keine Berührung
    case 0:
      bob3.setLed(EYE_1, OFF);
      break;

    // Arm 1 wird am oberen Sensor berührt
    case 1:
      bob3.setLed(EYE_1, KABARED);
      break;

    // Arm 1 wird in der Mitte berührt
    case 2:
      bob3.setLed(EYE_1, STEELBLUE);
      break;

    // Arm 1 wird am unteren Sensor berührt
    case 3:
      bob3.setLed(EYE_1, WHITE);
      break;
  }
}
```

In der Lerneinheit **acht** wird der **Anforderungsbereich II** umgesetzt: Die Schüler haben den existierenden Programmcode für Arm 1 getestet und verstanden und transferieren dies, um nun selbständig den Algorithmus für den Arm 2 zu programmieren. Sie verwenden eine neue Integer Variable **`wert2`** und speichern den Rückgabewert der Methode **`bob3.getArm(2)`** in dieser Variablen ab. Die folgende switch-case Struktur verzweigt dann anhand des jeweiligen Wertes der neuen Variablen.

Umsetzung Anforderungsbereich II

```ino
#include <BOB3.h>

void loop() {
  int wert1 = bob3.getArm(1);
  int wert2 = bob3.getArm(2);

  switch (wert1) {
    // keine Berührung
    case 0:
      bob3.setLed(EYE_1, OFF);
      break;
    // Arm 1 wird am oberen Sensor berührt
    case 1:
      bob3.setLed(EYE_1, KABARED);
      break;
    // Arm 1 wird in der Mitte berührt
    case 2:
      bob3.setLed(EYE_1, STEELBLUE);
      break;
    // Arm 1 wird am unteren Sensor berührt
    case 3:
      bob3.setLed(EYE_1, WHITE);
      break;
  }

  switch (wert2) {
    // schreibe hier den fehlenden Quellcode hin









  }
}
```



www.bob3.org

6. + 7. Unterrichtseinheit

In der sechsten und siebten Unterrichtseinheit beenden die Schülerinnen und Schüler das **„Intro III“**-Kapitel, sie bearbeiten die **Lerneinheiten neun**, **zehn** und **elf** und das Arbeitsblatt 12. Lernschwerpunkte sind hierbei **Funktionen**, Funktionen mit **Rückgabewerten**, Funktionen mit **Parametern** und die **Definition eigener Funktionen**. Die Schüler lernen, wie sie eine eigene Funktion definieren können, wie sie diese aufgerufen können und welche Möglichkeiten sich hieraus ergeben. Mittels kleiner Experimente werden **Blitzlichter** erzeugt, wobei eine Funktion mit Farben als Parametern bunte Blitzlichter erzeugt. Anhand der Software-Bibliothek von BOB3 werden die **Schlüsselworte void** und **int** bei Funktionendefinitionen erläutert und den Schülern wird der Unterschied zwischen **Funktionen** und **Methoden** verdeutlicht. Zum Abschluss der Lerneinheit erarbeiten die Schüler ein komplexeres Programm, das unter anderem einen eigenen **Blitzlicht-Gewitter-Generator** verwendet.

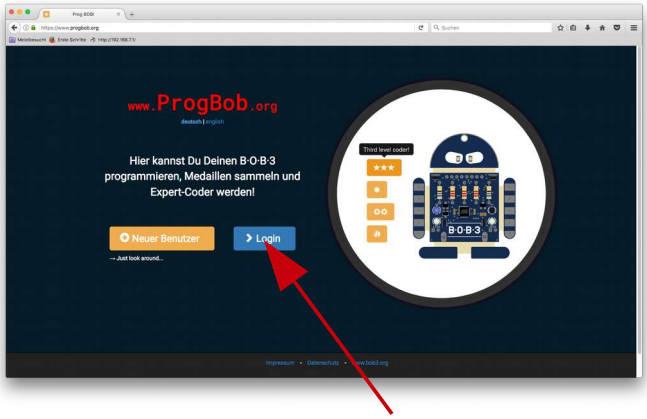
Zeitbedarf: ca. 90 min

Vorkenntnisse: BOB3 Modul 1+2

Material: Papier, Stift, Arbeitsblatt 12
PC/Laptop/Tablet, BOB3 mit Helm/Dock

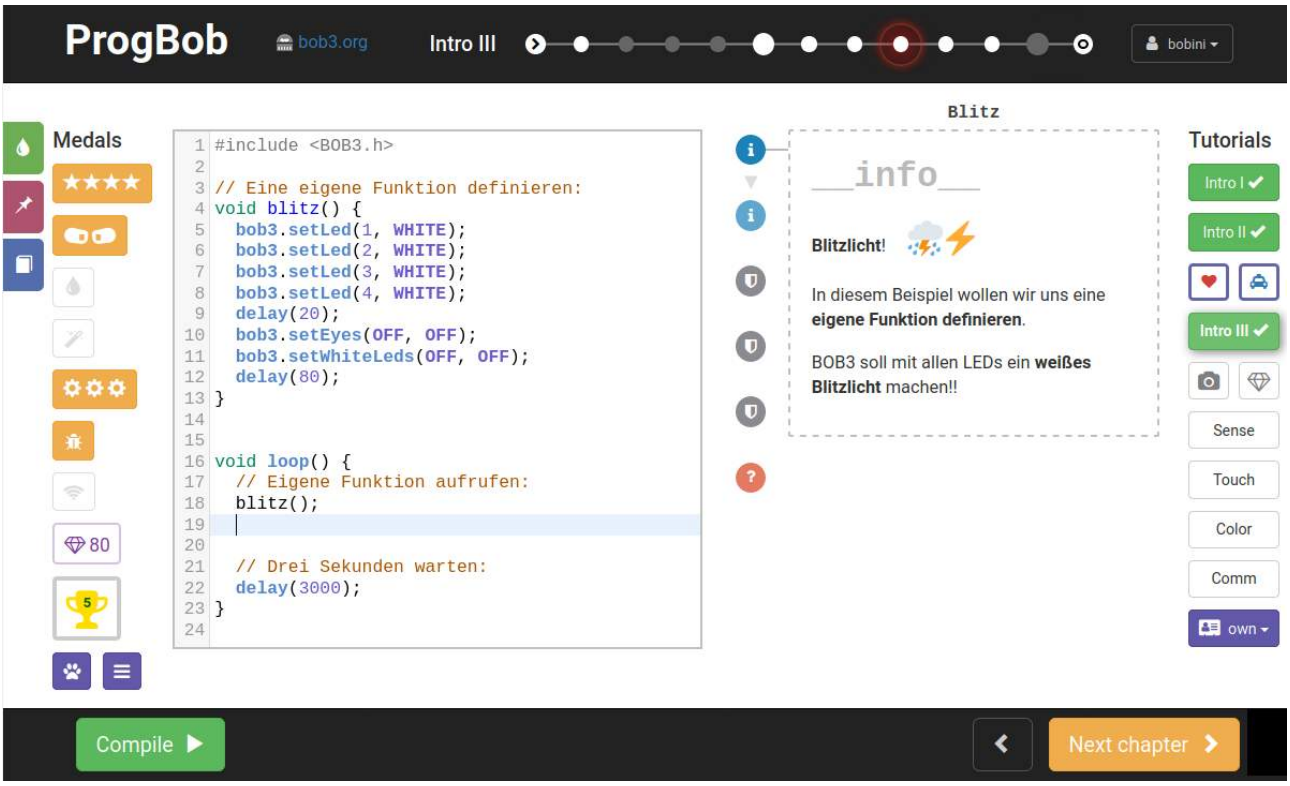
Ablauf

Die SuS starten den Webbrowser, gehen auf die Seite **http://www.ProgBob.org**, loggen sich mit Ihrem Account ein und gehen zum **„Intro-III“**-Kapitel:

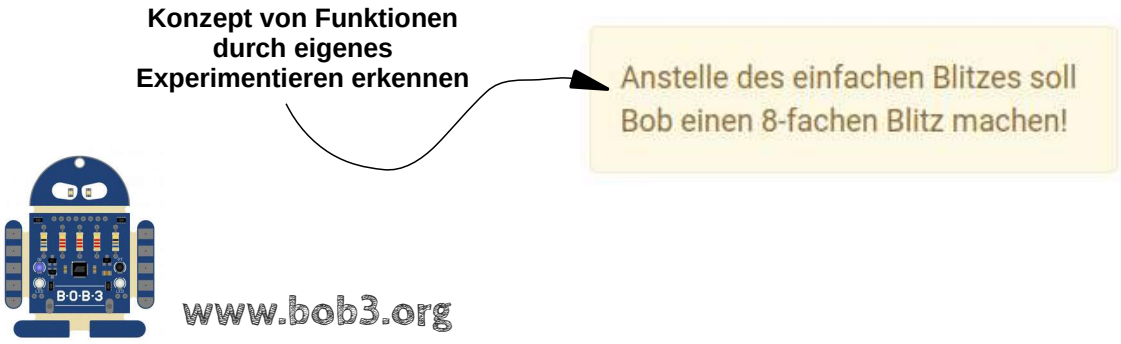


„Login“

Zunächst probieren die Schüler das bestehende Programm-Beispiel aus: BOB3 soll mit allen LEDs eins **weißes Blitzlicht** machen. Für dieses Blitzlicht wird eine **eigene Funktion** `blitz()` definiert, damit man anschließend ohne viel Tipparbeit auch z.B. ein **8-faches Blitzlicht** erzeugen kann. Die Schüler lernen, an welcher Stelle im Programm die Funktion definiert wird und wo und wie sie aufgerufen werden kann.



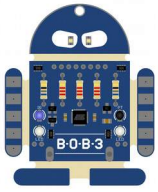
Die zugehörige **Wissensabfrage-Einheit** thematisiert die Frage, an welchen Stellen im Programm die Funktion aufgerufen werden kann. In der folgenden Lerneinheit vertiefen die Schülerinnen und Schüler ihr Wissen zu **Rückgabewerten** von Funktionen und vergleichen die schon bekannte Methode `bob3.getArm()` mit der neuen Funktion `blitz()` anhand der definierten Schlüsselwörter `int` und `void`. Die Schüler lernen, dass Funktionen, die sich auf ein Objekt beziehen, **Methoden** genannt werden und dass Bob's Funktionen Methoden heißen, da er im programmiertechnischen Sinne ein Objekt ist.



In der folgenden Lerneinheit definieren die Schüler eine neue Funktion **`blitzViolett()`**, die ein **violettes Blitzlicht** erzeugen kann und rufen diese innerhalb der **`loop()-Funktion`** auf. Das Lernziel hierbei ist zu vertiefen, dass Funktionen ohne Rückgabewert mit dem Schlüsselwort **void** definiert werden. Daher werden bei der Funktion **`blitzViolett()`** nur die definierten Anweisungen ausgeführt, der Aufruf der Funktion liefert keinen Wert zurück. Die anschließende Lerneinheit thematisiert **Funktionen mit Parametern**, die Schüler definieren eine neue Funktion **`blitzFarbe(int farbe)`**, die eine beliebige **Farbe** als Parameter übergeben bekommt. So lernen die Schüler in diesem Experiment, wie sie Funktionen mit Parametern richtig definieren und wie sie damit **Blitzlichter in allen Farben** erzeugen können.

```
#include <BOB3.h>


// Eigene Blitz-Funktion:
void blitzFarbe(int farbe) {
  bob3.setEyes(farbe, farbe);
  bob3.setWhiteLeds(ON, ON);
  delay(20);
  bob3.setEyes(OFF, OFF);
  bob3.setWhiteLeds(OFF, OFF);
  delay(80);
}


void loop() {

  delay(1000);
  blitzFarbe(ORANGE);
  blitzFarbe(RED);
  blitzFarbe(KABARED);
  blitzFarbe(ORANGE);

  delay(1000);
  blitzFarbe(ROYALBLUE);
  blitzFarbe(CYAN);
  blitzFarbe(BLUE);
  blitzFarbe(ROYALBLUE);

}

```

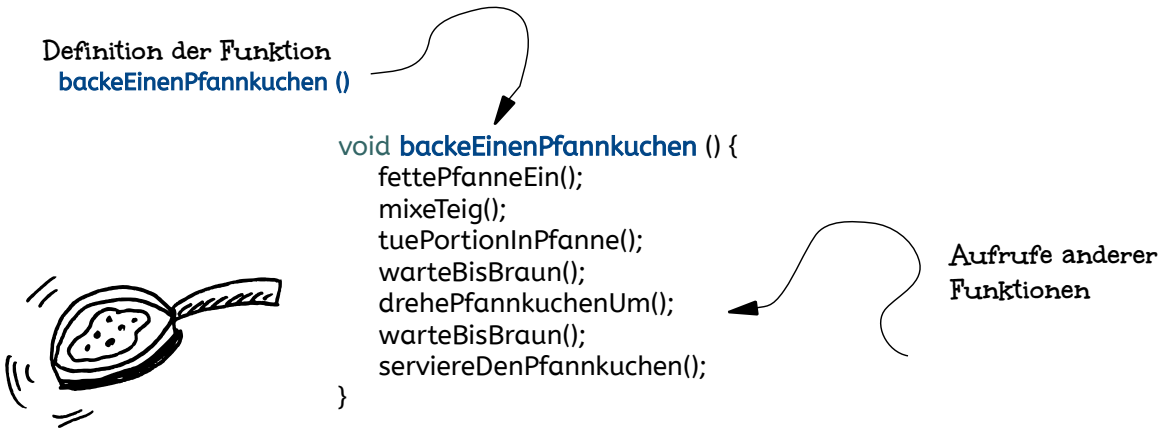
Um die neuen Konzepte weiter zu vertiefen vergleichen die Schüler in diesem Zusammenhang die neu definierten Funktionen mit bereits **bekannten** Funktionen und Methoden, wie z.B. der **`delay()`** Funktion oder der Methode **`bob3.getArm()`**. Hierbei wird deutlich, dass die Methode **`bob3.getArm()`** mit dem Schlüsselwort **`int`** definiert wird, also einen Integer Zahlenwert zurückliefert. Die Schüler sollen diskutieren, warum das Sinn macht und warum es sinnvoll ist, dass z.B. **`delay()`** als **`void`** definiert wird.



www.bob3.org

Die Schüler bearbeiten das **Arbeitsblatt 12** und lernen das Konzept und die Verwendung von Funktionen kennen:

„Was sind Funktionen?“



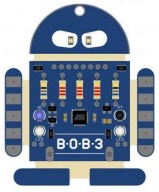
Die Schüler erlernen, dass Programme mittels **Funktionen** in einzelne **Teilbereiche/Teilprobleme** gegliedert werden können. Jede Funktion übernimmt eine bestimmte Aufgabe und kann selber auch wieder in verschiedene Teilbereiche zerlegt werden. Die Programme werden so strukturierter und übersichtlicher, zusätzlich ersparen Funktionen jede Menge Tipparbeit!

Dieses Konzept wird zunächst mit einem Beispiel aus der **Erlebniswelt** der Schüler vermittelt: Mal angenommen, man möchte einen **Pfannkuchen** backen, dann muss man einzelne Teilprobleme lösen: Man muss zuerst die Pfanne einfetten, dann den Teig mixen, eine Portion Teig in die Pfanne tun, kurz warten, bis der Teig braun ist, den Pfannkuchen wenden, wieder kurz warten und dann kann man den Pfannkuchen servieren! Für jedes dieser Teilprobleme definiert man sich eine eigene Funktion, z.B. die Funktion warteBisBraun ():

```
void warteBisBraun () {
    while (guckUnterPfannkuchen () != braun) {
        warte (1 min);
    }
}
```

Die Funktion warteBisBraun () löst die Teilaufgabe des eigentlichen Backvorgangs des Pfannkuchens: Man schaut unter den Pfannkuchen und solange die Funktion guckUnterPfannkuchen() zurückliefert, dass dieser noch nicht braun ist, wird eine Minute gewartet. In der Hauptfunktion backeEinenPfannkuchen() wird die Funktion warteBisBraun() dann zweimal aufgerufen, um den Pfannkuchen von beiden Seiten zu backen:

```
void backeEinenPfannkuchen ()
{
    fettePfanneEin();
    mixeTeig();
    tuePortionInPfanne();
    warteBisBraun();
    drehePfannkuchenUm();
    warteBisBraun();
    serviereDenPfannkuchen();
}
```



www.bob3.org

Die Schüler bearbeiten die Aufgaben 1-6 des Arbeitsblatts und besprechen ihre Lösungen. Aufgaben eins bis fünf thematisieren ‚Was ist eine Funktion?', ‚Welche Vorteile bietet eine Funktion?', ‚Welche der folgenden Funktionen/Methoden haben einen Rückgabewert?', ‚Ordne die folgenden Funktionen/Methoden aufsteigend anhand der Anzahl ihrer Parameter.' In der sechsten Aufgabe bekommen die Schüler die Arbeitsanweisung die im einleitenden Beispiel backeEinenPfannkuchen() verwendete Funktion mixeTeig() selber zu definieren:

Aufgabe 6: In unserer Funktion backeEinenPfannkuchen () von Blatt 1 wird unter anderem die Funktion mixeTeig () aufgerufen. Schreibe die Definition der Funktion mixeTeig (), verwende dazu folgende Funktionen, aber nur die passenden!!

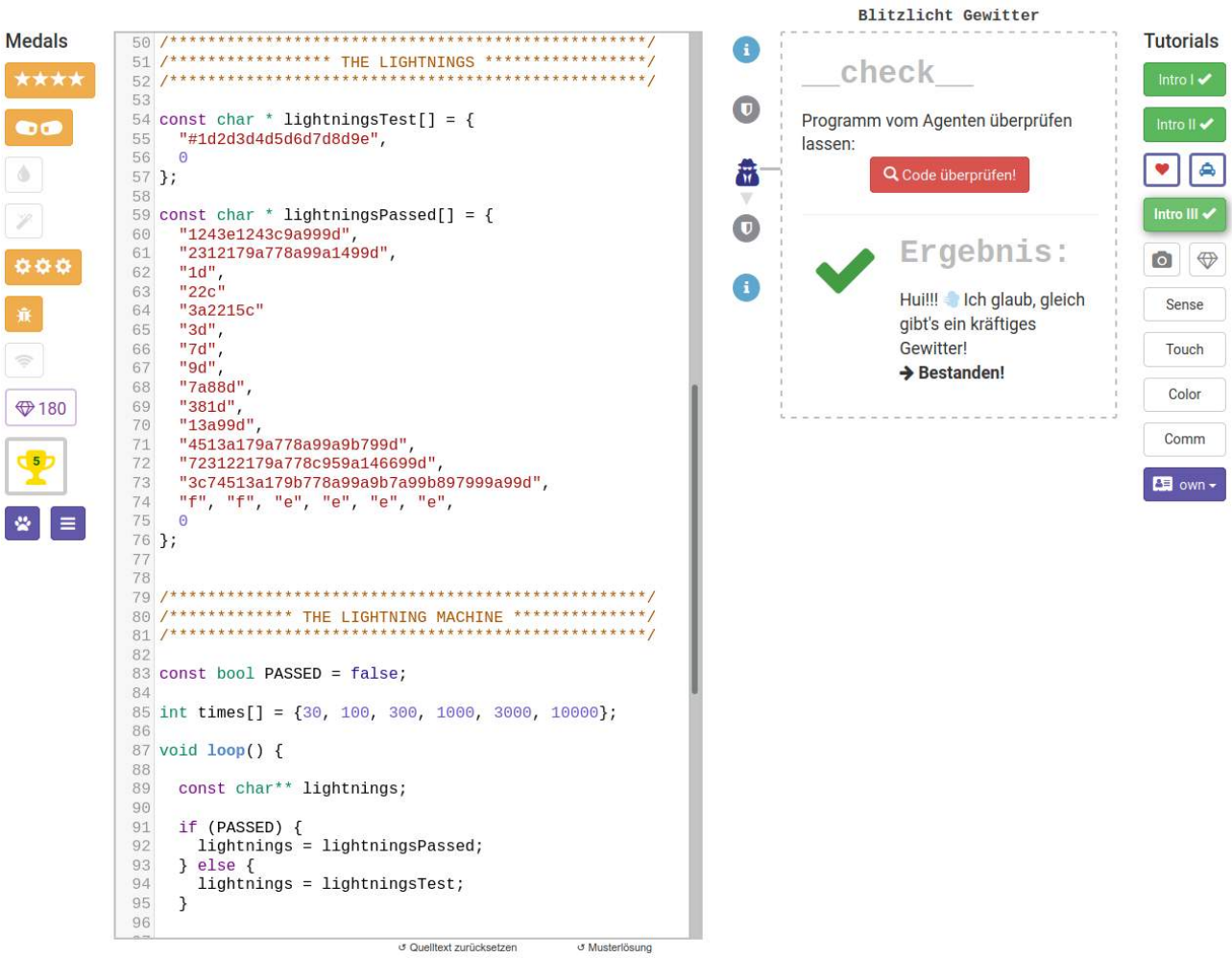
stelleSchuesselInDenOfen ()
fuegeMehlHinzu ()
stelleSchuesselBereit ()
quirlen ()
einePriseSalzHinein ()
fuegeEierHinzu ()
hackeKnoblauch ()
lassDieKatzeProbieren ()
fuegeSchokoladeHinzu ()
fuegeOSaftHinzu ()
mahleKaffeebohnen ()
fuegeZuckerHinzu ()
einSchussMineralwasserHinein ()
stelleKeksdoseBereit ()
fuegeMilchHinzu ()

Lerninhalte des „**Intro III**"-Kapitels:

- SuS intensivieren das Verständnis und die Verwendung von Variablen
- SuS entdecken experimentell weitere Sensoren, steuern diese an und werten die Daten aus:
  - Multifeld-Touch-Sensoren -> technisches Verständnis, Sensorwerte abfragen, abspeichern und verarbeiten, Zeit-Multiplex-Verfahren, Aufbau der Sensoren, Ansteuerung mittels Software-Bibliothek
  - IR-Sensor -> technisches Verständnis, Veranschaulichung des Messverfahrens mittels Oszilloskop, Sensorwerte auslesen, abspeichern und verarbeiten, Reflexions-Verfahren, Aufbau der Sensoren, Ansteuerung mittels Software-Bibliothek
- SuS lernen die switch/case Verzweigung als neue Kontrollstruktur kennen und anwenden
- SuS vertiefen die Kontrollstruktur ‚if/else' anhand einer Leistungsüberprüfung
- SuS lernen das Konzept von Funktionen kennen und unterscheiden Funktionen mit und Funktionen ohne Parameter
- SuS erlernen Funktionen mit und ohne Rückgabewert zu unterscheiden
- SuS verwenden Funktionen mit Rückgabewert mittels Sensorik
- SuS definieren eigene Funktionen und verstehen das resultierende Potential

Blitzlicht-Gewitter-Generator

Zum Abschluss der Lerneinheit programmieren die Schüler ein Programm, das einen **Blitzlicht - Gewitter-Generator** verwendet. Damit der Generator funktioniert, müssen die Schüler eine **switch-case-Verzweigung** implementieren: BOB3 soll zufällig in verschiedenen zeitlichen Abständen unterschiedlich starke Blitze erzeugen! Die Schülerlösung wird vom Agenten überprüft und sobald diese einwandfrei funktioniert, kann der Generator **aktiviert** werden:



8. Unterrichtseinheit

In der achten Unterrichtseinheit bearbeiten die Schüler das Kapitel „**Alarmanlage**“ und das Arbeitsblatt 13 - ‚**While-Schleife**‘. Sie beschäftigen sich in drei Lerneinheiten mit dem Konzept der While-Schleife als Struktur zur **wiederholten Durchführung** und lernen **Endlosschleifen** und das Konzept von **break-Anweisungen** kennen.

Zeitbedarf: ca. 45 min

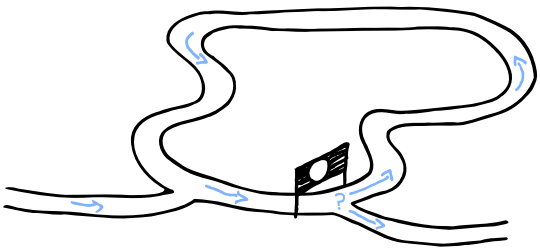
Vorkenntnisse: BOB3 Modul 1+2

Material: Papier, Stift, Arbeitsblatt 13
PC/Laptop/Tablet, BOB3 mit Helm/Dock

Ablauf

Die Schüler bearbeiten das **Arbeitsblatt 13** und lernen das Konzept und die Anwendungsmöglichkeiten von **while-Schleifen** kennen:

„Eine while-Schleife dient zur wiederholten Durchführung:“



Die SuS lernen die Kontrollstruktur **while-Schleife** kennen: Diese wird verwendet, um einen bestimmten Programmteil mehrfach zu **wiederholen**. Sie ermöglicht, dass in Abhängigkeit von einer **Bedingung** bestimmte Anweisungen **solange immer wieder** ausgeführt werden, bis die Bedingung **nicht mehr** erfüllt ist. Die Bedingung wird am Anfang der Schleife, also **vor** dem ersten Durchlauf geprüft. Falls die Bedingung schon bei der ersten Prüfung falsch ist, wird die Schleife gar nicht ausgeführt.

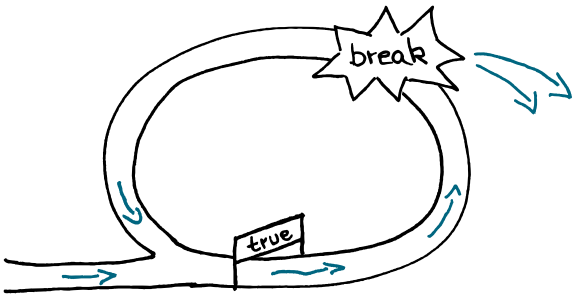
Was ist eine Endlosschleife?

Die Schüler lernen, dass eine **while-Schleife** auch als **Endlosschleife** verwendet werden kann und verstehen anhand eines Programmierbeispiels mit einem Warnblinklicht, wie sie diese programmieren können und wann dieses Konstrukt nützlich ist.

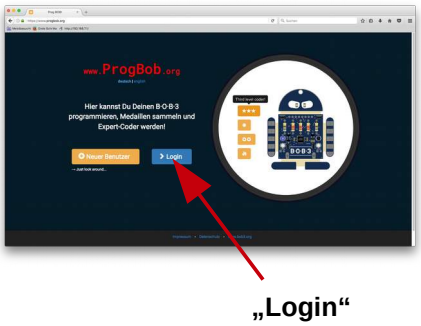
```
4
5   if (bob3.getIRSensor()>8) {
6     while (true) {
7       bob3.setWhiteLeds(ON, ON);
8       delay(200);
9       bob3.setWhiteLeds(OFF, OFF);
10      delay(200);
11    }
12  }
```

Sobald der Alarm **ausgelöst** wurde (Bedingung in Zeile 5 erfüllt), startet ein **Warnblinklicht** mit den weissen Bauch-Leds in einer **Endlosschleife**! Macht das Sinn? Ja, denn wenn der Alarm einmal ausgelöst wurde, dann soll Bob blinken und nicht mehr aufhören!

Die break-Anweisung



Im zweiten Teil des Arbeitsblatts lernen die Schüler die **break-Anweisung** kennen, die nicht nur bei while-Schleifen sondern auch in anderen Schleifen verwendet werden kann. Die break-Anweisung steht irgendwo innerhalb der Schleife und wird meistens in Kombination mit einer *if-Abfrage* verwendet. Sobald ein bestimmter Zustand eintritt und der Compiler bei der break-Anweisung ankommt, wird die Schleife **abgebrochen**. Anschließend starten die SuS den Webbrowser, gehen auf die Seite **http://www.ProgBob.org**, loggen sich mit Ihrem Account ein und gehen zum **„Intro-III“**-Kapitel:



„Login“

Anhand des Programmierbeispiels *‚Alarmanlage‘* vertiefen die Schüler das neu erlernte **Konzept** der **while-Schleife** zur wiederholten Durchführung. Die zu programmierende Alarmanlage startet in einer einfachen Version, die in den anschließenden Einheiten immer weiter, bis hin zu einer Alarmanlage mit einer **Reset-Funktion**, ausgebaut wird:

```
1  #include <BOB3.h>
2
3  void loop() {
4
5    if (bob3.getIRSensor() > 8) {
6
7      while (1 == 1) {
8        bob3.setWhiteLeds(ON, ON);
9        delay(200);
10       bob3.setWhiteLeds(OFF, OFF);
11       delay(200);
12     }
13   }
14 }
```

Alarmanlage Teil 1

__info__

Zur Programmierung verwenden wir eine neue **Programmier-Struktur**:

`while-Schleife`

- Eine **while-Schleife** prüft einen **Ausdruck** und wird **solange ausgeführt**, wie der Ausdruck **wahr** ist.
- Der Ausdruck wird dabei am **Anfang** der Schleife, also **vor** dem ersten Schleifendurchlauf ausgewertet.
- Eine **while-Schleife** kann mit einer `break;` Anweisung wieder verlassen, also gestoppt werden.

Die Schüler programmieren BOB3 so, dass er mit seinem IR-Sensor auf Objekte aufpassen kann und sofort bemerkt, wenn eine Hand die Objekte klaut. Im ersten Beispiel prüft die while-Schleife den Ausdruck 1 == 1, was immer true ergibt, weil es stimmt. Daher werden jetzt als Endlosschleife die Zeilen 8 bis 11 ausgeführt, bei BOB3 blinken jetzt für immer die beiden Bauch-Leds!

Macht das Sinn?

Die Schüler diskutieren die Sinnhaftigkeit der Aufgabenstellung und kommen zu dem Ergebnis, dass ein einmal ausgelöster Alarm nicht von selber wieder aufhören soll!

Im dritten Teil der Lerneinheit erweitern die SuS ihr Programm um eine Profi-Reset-Funktion, damit der jeweilige Besitzer der Objekte z.B. bei einem Fehlalarm den Bob in die Ausgangslage zurücksetzen kann. Hierbei wiederholen die SuS ihr Wissen über logische Operatoren.

__quiz__

1: Wie oft wird die Schleife `while(8 == 8)` ausgeführt?
- ○ keinmal
- ○ einmal
- ○ immer und immer wieder

2: Wie oft wird die Schleife `while(3 < 17)` ausgeführt?
- ○ keinmal
- ○ einmal
- ○ immer und immer wieder

3: Wie oft wird die Schleife `while(3 > 17)` ausgeführt?
- ○ keinmal
- ○ einmal
- ○ immer und immer wieder

Quiz auswerten!

Einbau einer Reset-Funktion

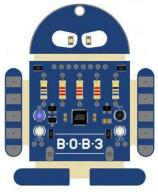
In der **dritten Ausbaustufe** des Programms bauen die Schüler eine Reset-Funktion ein, damit der Alarm bei Bedarf wieder **neu gestartet** werden kann. Nach Auslösung des Alarms blinkt BOB3 mit allen LEDs. Sobald dann beide Arme gleichzeitig berührt werden, wird das Blinken sofort ausgeschaltet. Nach einer zweisekündigen Pause wird der Alarm automatisch wieder scharfgeschaltet und Bob wartet auf den nächsten Dieb:

```
#include <BOB3.h>

void loop() {

  if (bob3.getIRSensor() > 8) {

    while (true) {
      bob3.setEyes(ORANGE, OFF);
      bob3.setWhiteLeds(ON, ON);
      delay(50);
      bob3.setEyes(OFF, OFF);
      bob3.setWhiteLeds(OFF, OFF);
      delay(50);
      bob3.setEyes(OFF, ORANGE);
      bob3.setWhiteLeds(ON, ON);
      delay(50);
      bob3.setEyes(OFF, OFF);
      bob3.setWhiteLeds(OFF, OFF);
      delay(50);

      // Profi-Reset-Funktion
      if ( (bob3.getArm(1)>0) && (bob3.getArm(2)>0) ) {
        break;
      }
    }
    delay(2000);
  }
}
```

Die Schüler lösen folgende Aufgabe: Falls Arm 1 und Arm 2 **gleichzeitig** berührt werden, dann soll das Blinken stoppen, die while-Schleife soll mittels einer **break-Anweisung** verlassen werden und anschließend soll Bob wieder in die Ausgangsposition zurückgesetzt werden. In der Umsetzung arbeiten die SuS mit einer **&&-Verknüpfung** der beiden **Sensor-Abfragen**:

```
(if ( (bob3.getArm(1)>0) && (bob3.getArm(2)>0) ){
   break;
}
```

Anschließend bearbeiten die Schüler die Aufgaben 1-6 des Arbeitsblatts und besprechen ihre Lösungen. Die Aufgaben eins bis vier thematisieren die **Anzahl der Durchläufe** verschiedener while-Schleifen. In der fünften Aufgabe wird ein konkretes Programm mit einer **break-Anweisung** dargestellt. Die SuS haben die Aufgabe zu entscheiden, welche Zeile nach dieser break-Anweisung ausgeführt wird. Die sechste Aufgabe des Arbeitsblatts thematisiert ein **Anwendungsbeispiel**. Die SuS sollen überlegen, was das Programm macht: ‚*Betrachte das folgende Programm. Für welchen Zweck könnte man es verwenden? Schreibe deine Ideen auf!*‘



www.bob3.org

9. Unterrichtseinheit

In der neunten Unterrichtseinheit vertiefen die Schülerinnen und Schüler die erlernten Konzepte mit dem „**Photo-Flash**“-Kapitel, das als **Anwendungsbeispiel** ausgelegt ist: Die Schüler programmieren BOB3 so, dass er den **Selbstauslöser** und das **Blitzlicht** eines Fotoapparats simuliert. Sobald beide Armsensoren gleichzeitig berührt werden, startet der Selbstauslöser gefolgt von einem Blitzlicht. Anhand dieses Beispiels aus der **Erlebniswelt** der Schüler werden die Programmierstrukturen ‚**if-else Verzweigungen**‘, ‚**for-Schleifen**‘, Arbeiten mit **Variablen**, mit **Funktionen** und **Rückgabewerten** vertieft. Zusätzlich wiederholen die SuS die Verwendung von **logischen Operatoren** und **Bedingungsprüfungen**.

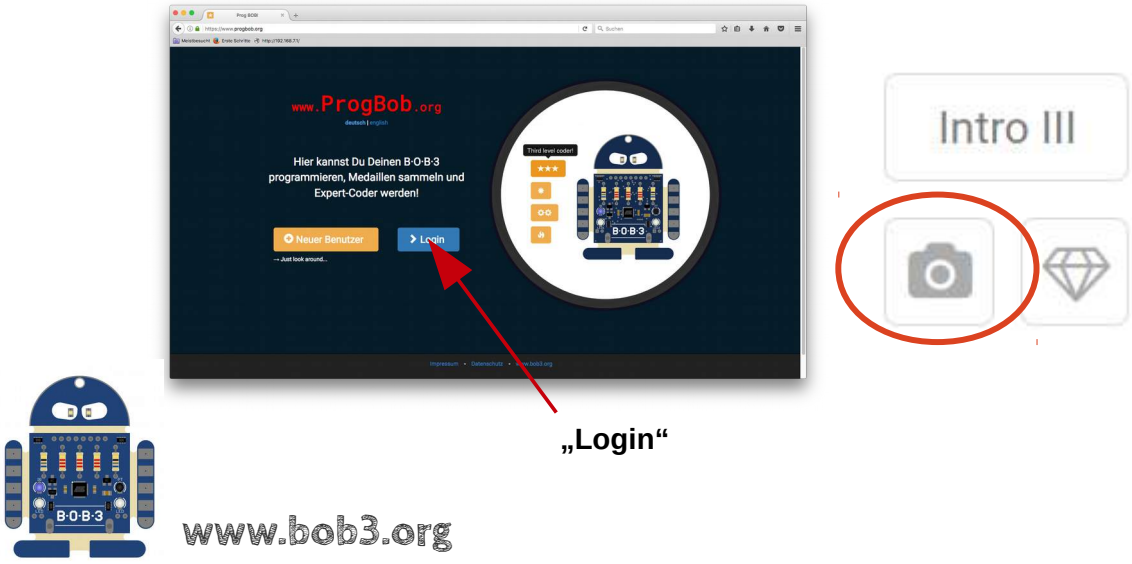
Zeitbedarf: ca. 45 min

Vorkenntnisse: BOB3 Modul 1+2

Material: PC/Laptop/Tablet, BOB3 mit Helm/Dock

Ablauf

Die SuS starten den Webbrowser, gehen auf die Seite **http://www.ProgBob.org**, loggen sich mit Ihrem Account ein und gehen zum Kapitel „**Photo-Flash**“:



In den folgenden Lerneinheiten entwickeln die Schüler **stufenweise** das Programm: Zunächst wird die Berührung von Armsensor 1 abgefragt und die Schüler speichern den Rückgabewert in einer Integer Variablen. Als Anzeige für Berührung/keine Berührung wird das Auge 1 ein- bzw. ausgeschaltet. Dies implementieren die Schüler mit einer if-else-Abfrage. Anschließend programmieren die Schüler selbstständig die Anzeige für Berührungen an Armsensor 2 und verwenden erneut die gelernten Programmierstrukturen.

```
1  #include <BOB3.h>
2
3  // Foto-Bob
4
5
6
7  void loop() {
8
9    // Arm 1 abfragen:
10
11
12
13
14
15   // Wenn Arm 1 berührt wird -> Auge 1 gelb an
16   if (      ) {
17
18   }
19
20
21   // Sonst -> Auge 1 aus
22   else {
23
24   }
25
26
27
28 }
29
```

Photo-Flash Teil 1

___aufgabe___

Programmiere die **Anzeige für Arm 1**:

Füge die folgenden Teile an den richtigen Stellen ein:

```
int wert1 = bob3.getArm(1);
```

```
wert1 != 0
```

Jetzt fehlen nur noch die Zeilen **17** und **23**:

```
??
```

```
??
```

Das kannst du schon, probiere mal!

Compiliere dein Programm und teste es mit BOB3!

Den **Selbstauslöser** implementieren die SuS mittels einer if-else-Verzweigung und der Verwendung von Vergleichsoperatoren und des logischen UND-Operators. Zur Vertiefung des Wissens zur Verwendung von Operatoren bearbeiten die Schüler anschließend eine passende Wissensabfrageeinheit.

(wahr) && (wahr) -> wahr
(wahr) && (falsch) -> falsch
(falsch) && (wahr) -> falsch
(falsch) && (falsch) -> falsch

___quiz___

1: Wahr oder falsch:
`(1 == 1) && (3 > 2)`
○ wahr ○ falsch

2: Wahr oder falsch:
`(3 == 3) && (5 < 2)`
○ wahr ○ falsch

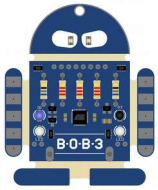
3: Wahr oder falsch:
`(4 != 5) && (199 > 201)`
○ wahr ○ falsch

4: Wahr oder falsch:
`(1024 <= 1022) && (9 >= 9)`
○ wahr ○ falsch

5: Wahr oder falsch:
`(8 != 0) && (33 > 32)`
○ wahr ○ falsch

6: Wahr oder falsch:
`(856 < 839) && (2 != 2)`
○ wahr ○ falsch

Quiz auswerten!



www.bob3.org

Im vierten Teil wird das Programm um ein **Aktivierungs-Blinken** ergänzt, um anzuzeigen, dass der Selbstauslöser gestartet wurde. In der **fünften** und letzten Lerneinheit erweitern die SuS ihr Programm um eine neue Funktion zur Simulation des Blitzlichts. Die Funktion **`blitz()`** wird mit den LEDs umgesetzt, die gleichzeitig **weiß aufblitzen**. Das **fertige Programm** sieht dann so aus:

```
#include <BOB3.h>

void blink() {
  for (int i=0; i<15; i++) {
    bob3.setEyes(WHITE, OFF);
    delay(50);
    bob3.setEyes(OFF, WHITE);
    delay(50);
  }
  bob3.setEyes(OFF, OFF);
}

void blitz() {
  bob3.setEyes(WHITE, WHITE);
  bob3.setWhiteLeds(ON, ON);
  delay(50);
  bob3.setEyes(OFF, OFF);
  bob3.setWhiteLeds(OFF, OFF);
}

void loop() {

  int wert1 = bob3.getArm(1);
  int wert2 = bob3.getArm(2);

  if ( wert1 != 0 ) {
    bob3.setLed(EYE_1, YELLOW);
  }
  else {
    bob3.setLed(EYE_1, OFF);
  }

  if ( wert2 != 0 ) {
    bob3.setLed(EYE_2, YELLOW);
  }
  else {
    bob3.setLed(EYE_2, OFF);
  }

  if ( (wert1 != 0) && (wert2 != 0) ) {
    bob3.setEyes(WHITE, WHITE);
    delay(500);
    // Aufruf blink()-Funktion:
    blink();
    delay(200);
    // Aufruf blitz()-Funktion:
    blitz();
    delay(2000);
  }
}
```

Definition der **Funktion blink()** → Implementation mittels **for-Schleife**

Definition der **Funktion blitz()** → Implementation mittels einer **50-Millisekunden Verzögerung**

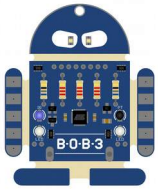
Deklaration der Variablen **wert1** und **wert2** zur Speicherung der Sensordaten

Auswertung der **Rückgabewerte** der Armsensoren mit einer **if-else-Struktur**

Implementation des Selbstauslösers → Umsetzung mittels **if-Abfrage** und einer Verknüpfung aus **Vergleichsoperatoren** und **logischen Operatoren**

Aufruf der **Funktion blitz()**

Aufruf der **Funktion blink()** mit anschließender Verzögerung



www.bob3.org

10., 11. und 12. Unterrichtseinheit

In der zehnten, elften und zwölften Unterrichtseinheit beschäftigen sich die Schülerinnen und Schüler vertieft mit dem **IR-Sensor** des Roboters. Das Arbeitsblatt 14 thematisiert den **Aufbau** und die **Funktion** des Sensors und behandelt insbesondere das verwendete **Reflexionsverfahren**. Die SuS lernen, dass der Sensor sowohl berührungslos **Objekte detektieren** und nah und fern unterscheiden kann, als auch als Detektor für **Tageslicht** verwendet werden kann. Diese verschiedenen Einsatzbereiche können anschaulich als **Sende- und Empfangssignale** des Sensors am **Oszilloskop** veranschaulicht werden (siehe Arbeitsblatt). Als Anwendungsbeispiele programmieren die SuS einen **Parksensor** als Assistenzsystem und einen **Tageslichtsensor**, der z.B. in Smart-Home-Anwendungen zum Einsatz kommen kann.

Zeitbedarf: ca. 135 min

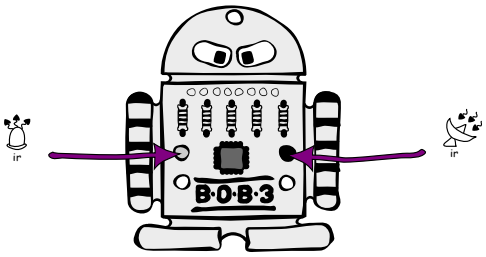
Vorkenntnisse: BOB3 Modul 1+2

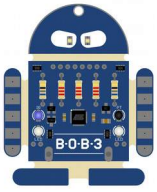
Material: Papier, Stift, Arbeitsblatt 14
PC/Laptop/Tablet, BOB3 mit Helm/Dock

Ablauf

Die Schüler bearbeiten das **Arbeitsblatt 14** und intensivieren das Verständnis für Aufbau, Funktion und programmiertechnische Ansteuerung des **IR-Sensors**:

„BOB3's Infrarotlicht-Sensor:"





Die Schüler lernen zunächst, dass der IR-Sensor aus zwei Teilen besteht, die verschiedene Aufgaben übernehmen: Eine **IR-Sende-LED** sendet Infrarotlicht aus und ein **IR-Empfänger** dient zur Detektion von IR-Licht.

www.bob3.org

Das Reflexionsverfahren

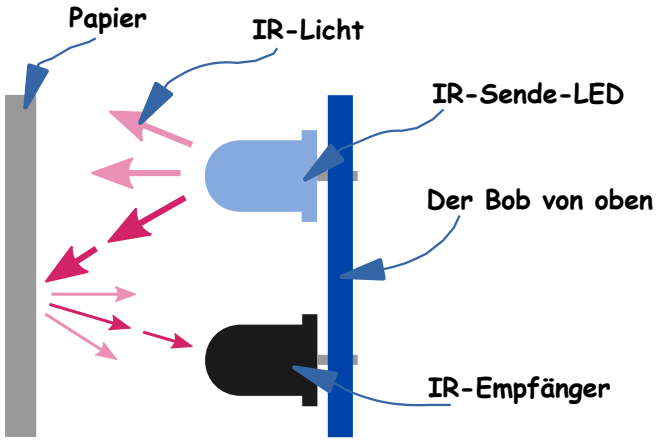
Die Schüler lernen, dass die **Detektion** von **nah** oder **fern** nach dem **Reflektionsverfahren** funktioniert: Die IR-Sende-LED sendet IR-Licht aus, dieses trifft dann auf ein Objekt oder ein Hindernis (z.B. ein Blatt Papier oder eine Hand), wird von dem jeweiligen Objekt **zurückreflektiert** und kann so von dem IR-Empfänger empfangen werden. **Je näher** das Papier vor dem Sensor ist, **desto mehr IR-Licht detektiert** der IR-Empfänger. Falls das Papier **weiter entfernt** vom Sensor ist, wird **wenig** reflektiertes IR-Licht detektiert.

Anhand des Anwendungsbeispiels ‚**Parksensor**' diskutieren und vertiefen die SuS das erlernte Reflexionsverfahren:

Fall 1) IR-Sensor detektiert kein Hindernis/Auto
→ *LEDs grün, freie Fahrt!*

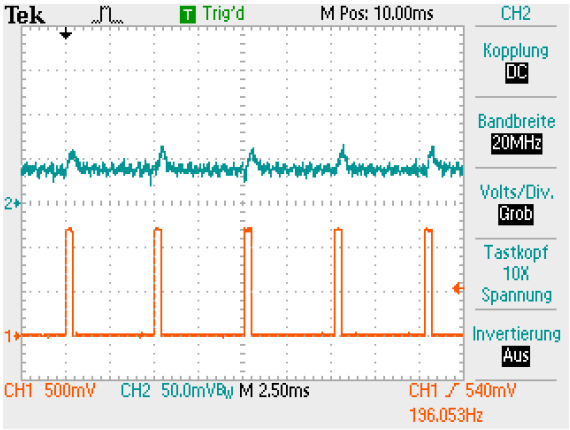
Fall 2) IR-Sensor detektiert ein etwas entferntes Hindernis/Auto
→ *LEDs orange, langsamer fahren und aufpassen!*

Fall 3) IR-Sensor detektiert ein nahes Hindernis/Auto
→ *LEDs rot, STOP!*



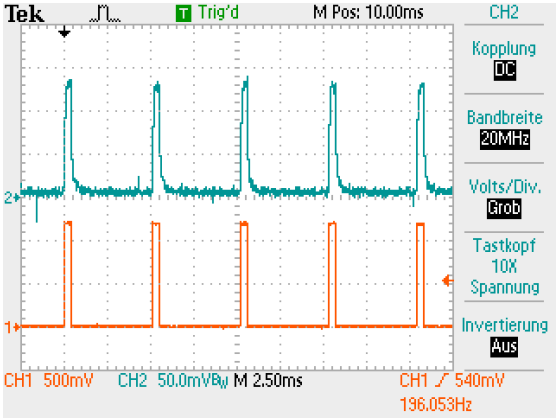
Messverfahren

Die **Signale** der IR-Sende-LED und des IR-Empfängers lassen sich sehr gut mit einem **Oszilloskop** veranschaulichen. In den folgenden Grafiken sieht man als obere Kurve das Signal des IR-Empfängers und als untere Kurve das Sende-Signal der IR-LED:

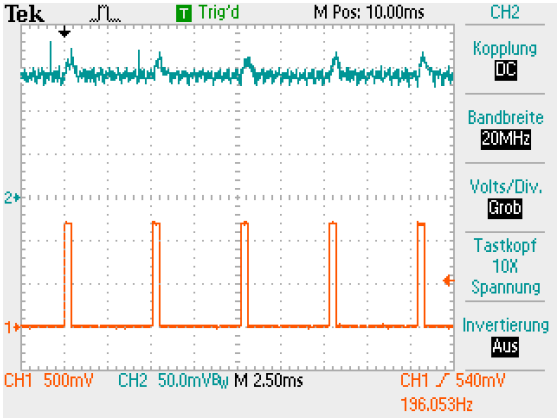


Szenario 1: *Keine Reflexion, kein Tageslicht*

Um das Messverfahren im Detail verstehen zu können, besprechen die Schüler die drei **Szenarien**: *– Keine Reflexion, kein Tageslicht – , – Reflexion an einer Hand, kein Tageslicht – und – Keine Reflexion, Tageslicht –*



Szenario 2: *Reflexion an einer Hand, kein Tageslicht*



Szenario 3: *Keine Reflexion, Tageslicht*

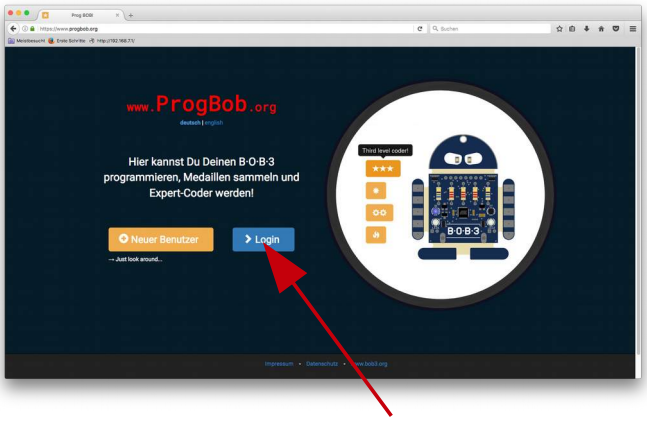
Die Schüler bearbeiten Aufgaben 1 – 4 + 6 des Arbeitsblatts und besprechen ihre Lösungen.

Software-Bibliothek

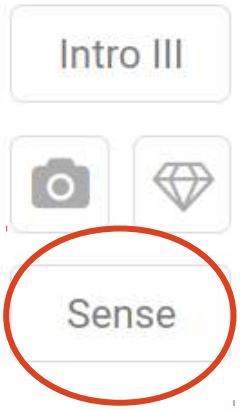
Zur **Ansteuerung** des IR-Sensors stehen in der Software-Bibliothek des BOB3 fertig implementierte Methoden zur Verfügung. Die Schüler beschäftigen sich mit der genauen **Bedeutung** der verschiedenen Methoden und deren **Rückgabewerten** und **Parametern**:

Methode	Bedeutung	Rückgabewert/Parameter
`bob3.getIRSensor()`	Liefert den Wert des IR-Reflexions-Sensors	0 - 255
`bob3.getIRLight()`	Liefert den Wert des IR-Umgebungslichts	0 – 255 255: direktes Sonnenlicht 0: Dunkelheit
`bob3.enableIRSensor(enable)`	Aktiviert bzw. deaktiviert den IR-Reflexions-Sensor	enable: 1 → aktiviert 0 → deaktiviert
`bob3.receiveMessage(timeout)`	Empfängt eine IR-Message	0 – 255 / -1 bei keinem Signal timeout: Millisekunden, die gewartet werden sollen
`bob3.transmitMessage(message)`	Sendet eine IR-Message	Message: 0 – 255, Zahl, die übertragen werden soll

Die SuS starten den Webbrowser, gehen auf die Seite **http://www.ProgBob.org**, loggen sich mit Ihrem Account ein und gehen zum Kapitel **„Sense“**:



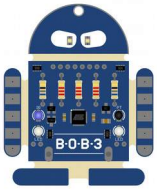
„Login“



Programmierung eines Parksensors

Die Lerneinheit enthält eine **offene Aufgabe** aus der **Erlebniswelt** der Schüler, um die bisher gelernten Konzepte und Strukturen zu vertiefen und anzuwenden: Die Schüler bekommen die Aufgabe, BOB3 so zu programmieren, dass er als **Parkassistenzsystem** verwendet werden kann.

```
1  #include <BOB3.h>
2
3  // B-O-B-3 als Einparkhilfe
4
5
6
7
8  void loop() {
9
10   // Integer Variable sensorWert deklarieren:
11
12   // aktuellen Sensor-Wert der Variablen zuweisen:
13
14
15
16
17
18
19
20
21
22 }
23
```



Die Schüler diskutieren zunächst, was ein Parkassistenzsystem ist und welche konkreten **Eigenschaften** solch ein System bietet. Anschließend entwickeln sie **Lösungsansätze** für die Umsetzung mit BOB3.

www.bob3.org

Offene Aufgabe – Parksensor

Die Schüler werden in Gruppen eingeteilt und erarbeiten konkrete Lösungsansätze für das Problem. Sie teilen das **Hauptaufgabe** in verschiedene zu lösende **Teilaufgaben** und entwickeln so Meilensteine, die die Gruppen jeweils individuell umsetzen:

```
#include <BOB3.h>

// B-O-B-3 als Einparkhilfe

// Neue Funktion: Warnblitzlicht
void blitzen() {
  bob3.setWhiteLeds(ON, ON);
  delay(20);
  bob3.setWhiteLeds(OFF, OFF);
}

void loop() {

  // Integer Variable sensorWert deklarieren:
  int sensorWert;

  // aktuellen Sensor-Wert der Variablen zuweisen:
  sensorWert = bob3.getIRSensor();

  // Hindernis ganz nah -> Augen rot, zu nah!
  if (sensorWert > 10) {
    bob3.setEyes(RED, RED);
    blitzen();
  }

  // Hindernis in Sicht -> Augen unicorn, Abstand ok!
  else if(sensorWert > 4) {
    bob3.setEyes(UNICORN, UNICORN);
  }

  // kein Hindernis in Sicht -> Augen grün, freie Fahrt!
  else{
    bob3.setEyes(FORESTGREEN, FORESTGREEN);
  }

  delay(100);
}
```

Meilenstein 4:

Implementation einer neuen Funktion blitzen(), ohne Parameter und ohne Rückgabewert, Aufruf der Funktion innerhalb der loop()-Funktion

Meilenstein 1:

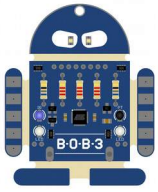
Deklaration einer Integer Variablen, Abfrage des aktuellen Sensorwerts mittels der Methode bob3.getIRSensor() und Speicherung des Rückgabe-werts der Methode in der Variablen

Meilenstein 2:

Implementation einer if-else-Struktur zur Unterscheidung der Fälle
1) Hindernis zu nah
2) kein Hindernis
Verwendung der Variablen mittels Vergleichsoperatoren

Meilenstein 3:

Erweiterung der if-else-Struktur zur Unterscheidung eines dritten Falls:
1) Hindernis zu nah
2) kein Hindernis
3) Hindernis in Sicht, Abstand in Ordnung



www.bob3.org

Smart-Home Anwendung – Tageslichtsensor

Die Lerneinheit ‚**Programmierung eines Tageslichtsensors**' enthält die Aufgabe, BOB3 so zu programmieren, dass er mithilfe seines IR-Sensors detektiert, ob er sich im **Dunkeln** oder im **Tageslicht** befindet. Zunächst lernen die Schüler, welche verschiedenen **Lichtarten** es gibt und wie diese sich unterscheiden:

Es gibt verschiedene **Lichtarten**:

- Infrarotes-Licht (IR)
- Sichtbares Licht
- Ultraviolettes-Licht (UV)

__info__

Jetzt wollen wir BOB3 als **Tageslicht-Sensor** programmieren!

Er soll mithilfe seiner **IR-Sensorik** detektieren, ob er sich im **Dunkeln** oder im **Tageslicht** befindet.

Sobald die Sensoren feststellen, dass es dunkel ist, sollen alle LEDs weiß leuchten!

Die Schüler lernen, dass **Tageslicht** aus IR-Licht, also aus sichtbarem Licht und aus UV-Licht besteht. **Glühlampen** haben einen IR-Anteil, einen sichtbaren Anteil und einen UV-Anteil. **LED-Lampen** und **Neonröhren** haben dagegen nur einen sichtbaren Anteil! BOB3 detektiert also mit dem IR-Sensor den IR-Anteil im Tageslicht und kann so dunkel und hell unterscheiden. Sobald der Sensor feststellt, dass es dunkel ist, sollen alle LEDs eingeschaltet werden. Dies ist eine typische Smart-Home Anwendung einer **automatischen Beleuchtung**.

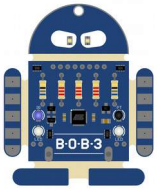
Die Schüler bearbeiten die Aufgabe 5 des Arbeitsblatts und besprechen ihre Lösungen.

Zur Programmierung verwenden die SuS die Funktion `bob3.getIR-Light()` und speichern den abgefragten Sensorwert in einer Integer Variablen. Mittels einer if-else Abfrage wird die Auswertung implementiert: Je nach Sensorwert werden die LEDs ein- oder ausgeschaltet.

BOB3 mit seinem IR-Sensor als Tageslicht-Sensor reagiert also auf den IR-Anteil im Tageslicht!

```
bob3.getIRLight()
```

```
#include <BOB3.h>

// B-O-B-3 als Tageslicht-Sensor

void loop() {

  // aktuellen IR-Licht Wert abfragen
  int irWert;
  irWert = bob3.getIRLight();

  if (irWert < 10) {
    bob3.setEyes(WHITE, WHITE);
    bob3.setWhiteLeds(ON, ON);
  } else {
    bob3.setEyes(OFF, OFF);
    bob3.setWhiteLeds(OFF, OFF);
  }

  delay(100);
}
```



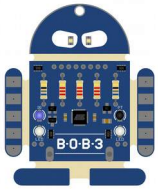
www.bob3.org

Differenzierung – Grenzwerte manuell setzen

Zur weiteren Differenzierung kann das Programm von den Schülern so **erweitert** werden, dass die **Lichtempfindlichkeit** über das Berühren von Arm 2 gesteuert wird:
Die Schüler deklarieren eine **globale Variable** `grenzWert` und initialisieren diese mit dem Wert 10. Immer, wenn Arm 2 oben berührt wird, soll die Variable `grenzWert` um 10 erhöht werden. Wenn Arm 2 unten berührt wird, soll die Variable `grenzWert` jeweils um 10 vermindert werden.

```
#include <BOB3.h>

// B-O-B-3 als Tageslicht-Sensor

// Grenzwert festlegen
int grenzWert = 10;

void loop() {
  // aktuellen IR-Licht Wert abfragen
  int irWert = bob3.getIRLight();

  // Weniger Licht als der Grenzwert -> Lampen an
  if (irWert < grenzWert){
    bob3.setEyes(WHITE, WHITE);
    bob3.setWhiteLeds(ON, ON);
  }

  // Mehr Licht vorhanden -> Lampen bleiben aus
  else {
    bob3.setEyes(OFF, OFF);
    bob3.setWhiteLeds(OFF, OFF);
  }

  // Falls Arm 2 oben berührt wird
  // Variable grenzWert um 10 erhöhen:
  if (bob3.getArm(2) == 1) {
    grenzWert = grenzWert + 10;
  }

  // Falls Arm 2 unten berührt wird
  // Variable grenzWert um 10 vermindern:
  if (bob3.getArm(2) == 3) {
    grenzWert = grenzWert - 10;
  }

  delay(100);
}

```

Die Schüler testen ihre implementierte Lösung und diskutieren, warum die Variable `grenzWert` als **globale** Variable und nicht als **lokale** Variable definiert wird. Abschließend bearbeiten die Schüler die passende Wissensabfrageeinheit.



www.bob3.org