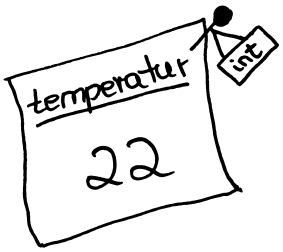
Arbeitsblatt 9 – Variablen

Was ist eine Variable?



Eine Variable ist ein Speicherort für *Zahlen*, *Zeichen* oder *sonstige Daten*. Wir werden zunächst nur mit Zahlen arbeiten und merken uns daher erstmal: **Eine Variable ist ein Speicherort für Zahlen.**

Jede Variable hat einen **Namen** (den suchst du selber aus) und einen **Datentyp**. Wir verwenden in unseren Beispielen den Datentyp **Integer** für **ganze Zahlen**. Wenn man z.B. für ein Programm eine Variable benötigt, die ganzzahlige Temperaturwerte abspeichern kann, dann schreibt man:

Wir könnten die Variable auch anders nennen, z.B. **int temp;** oder auch **int Temperatur;** oder auch **int temp1;** . Das Schlüsselwort für ganze Zahlen muss immer **int** heissen.

Wofür verwendet man Variablen?

Das folgende Programm verwendet unsere Variable **temperatur**. Hast du eine Idee, was das Programm macht? Überlege mal!

```
1  #include <BOB3.h>
2
3  void setup() {
4
5    int temperatur = 22;
6
7    if ( temperatur < 5 ) {
8      bob3.setEyes(BLUE, BLUE);
9    }
10   else if (temperatur < 15) {
11     bob3.setEyes(YELLOW, YELLOW);
12   }
13   else if (temperatur < 25) {
14     bob3.setEyes(ORANGE, ORANGE);
15   }
16   else {
17     bob3.setEyes(RED, RED);
18   }
19
20 }
```

In Zeile 5 wird eine Integer-Variable mit dem Namen temperatur deklariert, also neu eingeführt und mit der Zahl 22 initialisiert. Initialisieren bedeutet, dass die Variable einen Anfangswert zugewiesen bekommt. Ab jetzt ist das Wort **temperatur** gleichbedeutend mit der Zahl **22**. Daher können wir im folgenden Programm, also ab Zeile 7, mit der Variablen **temperatur** arbeiten.

Bei der Abfrage **if (temperatur < 5)** wird also **22 < 5** ausgewertet.

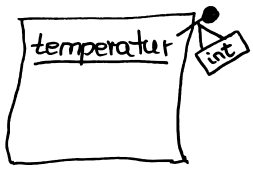
Was macht BOB3, wenn du das Programm ausprobierst?
Was ändert sich, wenn wir in Zeile 5 den Wert der Variablen von der **22** in eine **30** ändern?
Was macht Bob dann?

Variablen sind variabel!

Da Variablen **variabel** sind, kann man den jeweils gespeicherten **Wert** beliebig **ändern**. Der Datentyp muss dabei immer gleich bleiben: In einer Integer Variablen können nur ganze Zahlen gespeichert werden! Wenn man z.B. eine Kommazahl abspeichern möchte, dann muss man einen anderen Datentyp verwenden!

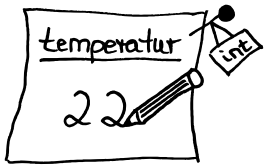
Die Arbeit mit Variablen kann man sich in etwa so vorstellen:

int temperatur; ⟷



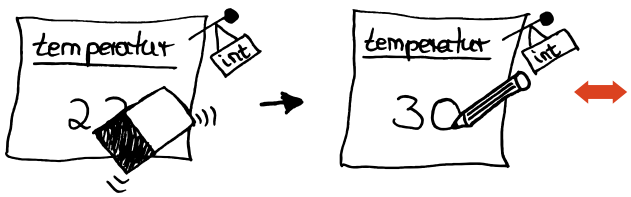
⟷ *Es wird eine Variable namens temperatur vom Typ Integer deklariert, also neu angelegt.*

temperatur = 22; ⟷



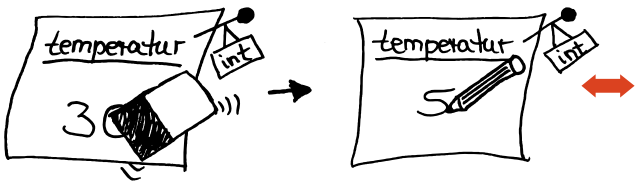
⟷ *Der Variablen temperatur wird die Zahl 22 zugewiesen.*

temperatur = 30; ⟷



⟷ *Der Variablen temperatur wird der Wert 30 zugewiesen, d.h. der Wert der Variablen wird auf 30 geändert.*

temperatur = 5; ⟷



⟷ *Der Variablen temperatur wird der Wert 5 zugewiesen.*

Aufgabe 1: Beschreibe, was eine Variable ist und welche Eigenschaften sie hat.

Aufgabe 2: Beschreibe, was das Programm von Blatt 1 macht.

Aufgabe 3: Warum heisst eine Variable ‚Variable'? Erläutere deine Antwort!

Aufgabe 4: Betrachte das folgende Beispiel. Welchen Wert hat die Variable **temp** nach der Zeile 7?

```
5  int temp = 5;
6  temp = 8;
7  temp = temp + 4;
```

Aufgabe 5: Welche Programm-Codes sind korrekt und würden compilieren? Kreuze die richtigen Antworten an, es sind mehrere Antworten möglich:

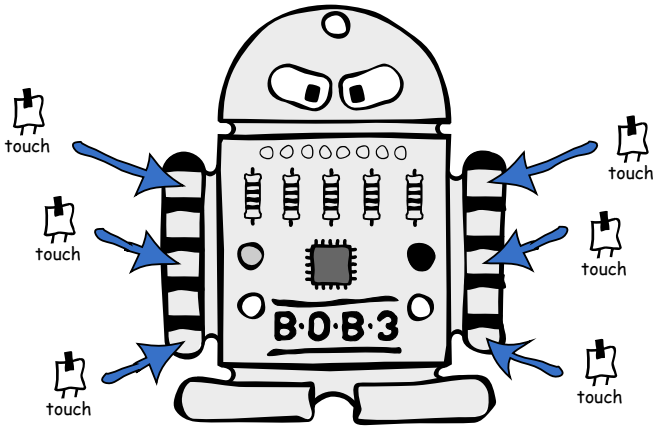
- ☐ **`int temperatur == 22;`**
- ☐ **`int Temperatur = 22;`**
- ☐ **`int temperatur = 22;`**
- ☐ **`int temp = 22,`**
- ☐ **`int temp = 22;`**
- ☐ **`int temp22 = 22;`**
- ☐ **`integer temperatur = 22;`**
- ☐ **`Integer temperatur = 22;`**
- ☐ **`int temperatur = 11;`**
- ☐ **`int temp = 11;`**

Aufgabe 6: Welche Aussagen sind richtig? Kreuze die richtigen Aussagen an:

- ☐ Eine Variable kann gleichzeitig mehrere Zahlen abspeichern
- ☐ Eine Variable kann immer nur genau eine Zahl abspeichern
- ☐ Eine Variable kann auch Zeichen abspeichern
- ☐ Der Wert einer Variablen kann verändert werden
- ☐ Variablen vom Typ Integer speichern ganze Zahlen
- ☐ Variablen vom Typ Integer speichern Kommazahlen
- ☐ Variablen vom Typ Integer speichern Zeichen
- ☐ Eine Variable behält immer ihren Initialwert
- ☐ Eine Variable hat immer einen Namen und immer einen Datentyp
- ☐ Eine Variable hat immer einen Namen und manchmal einen Datentyp
- ☐ Der Datentyp bestimmt die Art der zu speichernden Daten

Arbeitsblatt 10 - Touch-Sensoren

BOB3's Arme sind Multifeld-Touch-Sensoren



Beide Arme vom BOB3 sind **Multifeld-Touch-Sensoren**. Die Arme „merken", ob sie berührt werden oder nicht! Weil der Bob sogar bemerkt, **wo** du den jeweiligen Arm berührst, ob oben, mittig oder unten, sind es Multifeld-Touch-Sensoren. Bob hat damit insgesamt **sechs Tastsensoren**, die du ansteuern oder abfragen kannst!

Durch die Unterscheidbarkeit zwischen **oben**, **mittig** und **unten** kann man Programme schreiben, die die sechs Sensoren zur Ansteuerung verschiedener Aktivitäten verwenden, z.B. um Spiele zu programmieren oder bestimmte Verhaltensweisen des Roboters zu **starten**. Ein codiertes Programm könnte z.B. erst dann starten, wenn der Roboter mittels einer bestimmten Kombination der Armsensoren **entsperrt** wurde, oder man verwendet eine andere Kombination dazu, den Roboter in einen **gesperrten** Zustand zu versetzen!

Zeit-Multiplex-Verfahren

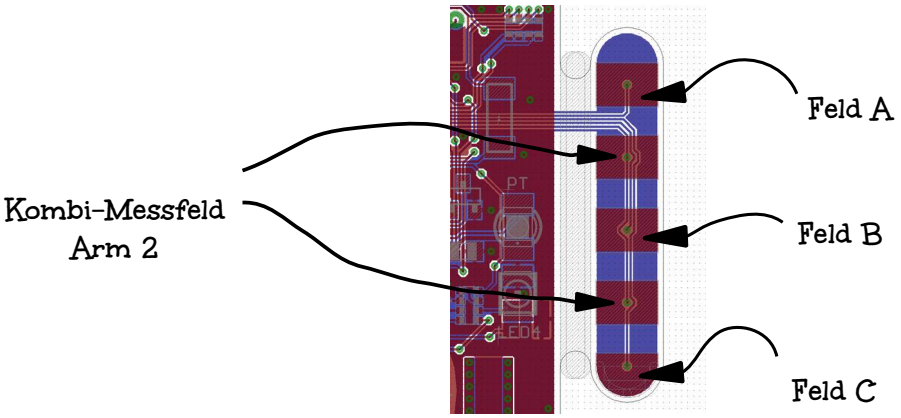
Bob's Touch-Sensoren arbeiten im **Zeit-Multiplex-Verfahren**, dies ist eine Methode zur Signalübertragung, bei der mehrere Signale zusammengefasst werden. So kann man viele Signale an wenigen Mikrocontroller-Eingängen anschließen. Dabei teilen sich mehrere Signale einen Eingang: Der Controller von BOB3 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**!

Der Trick ist das Zusammenspiel aus Aktivierungs- und Messfeldern. Von den drei Aktivierungsfeldern (A (oben), B (mittig) oder C (unten)) wird jeweils eins eingeschaltet und dann werden an beiden Armen die beiden Messfelder gemessen. Der Mikrocontroller sendet z.B. ein Signal an das

Aktivierungsfeld A und misst danach, ob er dieses Signal über das Kombi-Messfeld zurückempfängt. Falls in dem Moment Arm 1 oben berührt wird, hat der Controller einen Rückempfang und 'bemerkt' so, dass gerade das Messfeld und das Aktivierungsfeld A über einen Finger kurzgeschlossen werden. Nachdem alle drei Aktivierungsfelder aktiviert wurden haben wir sämtliche 6 Ergebnisse. Dabei interessiert uns nur die Ja/Nein Information, d.h. ob die Felder berührt werden oder nicht. Der Controller von BOB3 macht das ca. 200 mal pro Sekunde!!!

Aufbau der Sensoren

Jeder Arm besteht jeweils aus fünf Feldern: 3 **Aktivierungsfelder** (A, B, C) und 2 **Messfelder**. Sobald du ein *Aktivierungsfeld* **gleichzeitig** mit einem *Messfeld* berührst, bekommt der Bob ein Signal, ob Feld A, Feld B oder Feld C berührt wurde.



Die beiden Rechtecke des Kombi-Messfelds sind elektrisch miteinander verbunden!

Software-Bibliothek

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:

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

id	1	2
Sensor	Arm 1	Arm 2

Mit der Methode `bob3.enableArms(ARMS_DETECTOR)` können die Sensoren z.B. für den **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.

Aufgabe 1: Beschreibe, wie das Multiplexverfahren funktioniert.

Aufgabe 2: Beschreibe, aus welchen Teilen die Armsensoren von BOB3 bestehen und wie sie funktionieren.

Aufgabe 3: Welche Programm-Codes sind korrekt und würden compilieren? Kreuze die richtigen Antworten an, es sind mehrere Antworten möglich:

- ☐ **`int sensorWert = bob3.getArm();`**
- ☐ **`int sensorWert == bob3.getArm(2);`**
- ☐ **`int sensorWert = bob3.getArm(1,2);`**
- ☐ **`int sensorWert = bob3.getArm(2,1);`**
- ☐ **`int wert = bob3.getArm(2,1);`**
- ☐ **`int WERT = bob3.getArm(2);`**
- ☐ **`int wert = getArm(1);`**
- ☐ **`if (2 == bob3.getArm(1))`**
- ☐ **`if (bob3.getArm(2) == 1)`**
- ☐ **`int sensorWert = bob3.getArm(1);`**
- ☐ **`int sensorWert = bob3.getArm(2);`**

Aufgabe 4: Betrachte das folgende Programm und beschreibe genau, was bei welcher Aktion am Bob passiert!

```
#include <BOB3.h>

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

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

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

  if ((wert1 == 2) && (wert2 == 2)) {
    bob3.setEyes(ORANGE, WHITE);
    bob3.setWhiteLeds(ON, OFF);
    delay(200);
    bob3.setEyes(WHITE, ORANGE);
    bob3.setWhiteLeds(OFF, ON);
    delay(200);
  }

}

```

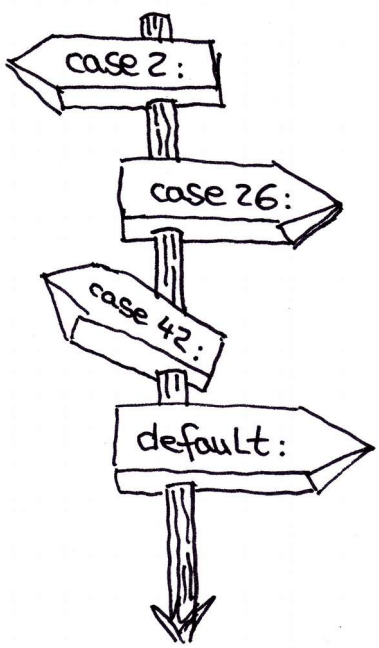
Aufgabe 5: Nenne alle möglichen Rückgabewerte der Methode `bob3.getArm()`.

Arbeitsblatt 11 - switch-case

Eine switch-case-Struktur ist eine Verzweigung

Die Kontrollstruktur **switch-case** ist eine Verzweigung, die dazu dient **viele verschiedene Fälle** zu unterscheiden.

Für jeden Fall werden unterschiedliche Anweisungen ausgeführt. Die Kontrollstruktur ermöglicht, dass in Abhängigkeit vom aktuellen Wert einer **Variablen** oder eines Ausdrucks bestimmte Anweisungen ausgeführt werden und andere dagegen nicht!




```
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;
}
```

Die Kontrollstruktur wird mit dem Schlüsselwort **switch** eingeleitet. In den folgenden runden Klammern steht die **Variable**, die ausgewertet werden soll.

Anschließend werden die verschiedenen Fälle aufgeführt: Jeder Fall beginnt mit dem Schlüsselwort **case**, dem jeweiligen Wert und einem Doppelpunkt. Dann folgen die auszuführenden Anweisungen. Jeder Fall wird mit einer **break-Anweisung** abgeschlossen!

Je nach Wert der Variablen wird der entsprechende case-Zweig ausgewählt. Daraufhin werden die Anweisungen dieses Zweigs ausgeführt. Mit der anschließenden break-Anweisung wird der Zweig und die komplette switch-case-Verzweigung beendet.

Falls kein case erreicht wird, also kein passender Fall vorhanden ist, so werden die Anweisungen des **default-Zweigs** ausgeführt.

Wir schauen uns mal ein konkretes Beispiel an:

```
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 }
```

Das Programm enthält eine **switch-case-Verzweigung**, die anhand des Werts der Variablen `anzahlLampen` verschiedene Anweisungen ausführt: BOB3 soll genau die Anzahl an LEDs einschalten, die die Variable `anzahlLampen` vorgibt.

Im Beispiel ist `anzahlLampen` auf `3` gesetzt, daher wird der Zweig `case 3:` ausgeführt: es werden also 3 LEDs eingeschaltet:

```
case 3:
  bob3.setEyes(WHITE, WHITE);
  bob3.setWhiteLeds(ON, OFF);
  break;
```

Am Bob werden die beiden Augen-LEDs und eine Bauch-LED, also insgesamt 3 LEDs eingeschaltet. Mit der abschließenden **break-Anweisung** wird der Zweig und die komplette switch-case-Verzweigung verlassen!

Aufgabe 1: Beschreibe, was die switch-case-Kontrollstruktur ist und wann man sie anwendet.

Aufgabe 2: Werden bei der switch-case-Verzweigung des Beispiels von Blatt 2 **alle** Fälle ausgeführt? Begründung!

Aufgabe 3: Was passiert am BOB3, wenn im Beispiel von Blatt 2 in Zeile 5 die **drei** in eine **eins** geändert wird: `int anzahlLampen = 1;` ?
Welcher Zweig wird ausgeführt?

Aufgabe 4: Was passiert am BOB3, wenn im Beispiel von Blatt 2 in Zeile 5 die **drei** in eine **fünf** geändert wird: `int anzahlLampen = 5;` ?
Welcher Zweig wird ausgeführt?

Aufgabe 5: 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
```

Aufgabe 6: Betrachte das Programm von Aufgabe 5. Was passiert, wenn Bob‘s Arm 2 mittig berührt wird? Lies die Aufgabenstellung genau!

- ☐ Beide Augen leuchten orange
- ☐ Beide Augen leuchten weiss
- ☐ Beide Augen leuchten nicht
- ☐ Beide Augen leuchten grün
- ☐ Beide Augen leuchten rot
- ☐ Beide Bauch-LEDs leuchten

Aufgabe 7: Betrachte das Programm von Aufgabe 5. Welche Zeile wird **nach** der break-Anweisung in Zeile 25 ausgeführt?

- ☐ Zeile 9
- ☐ Zeile 27
- ☐ Zeile 11
- ☐ Zeile 33
- ☐ Zeile 22
- ☐ Zeile 34

Arbeitsblatt 12 - Funktionen

Was sind Funktionen?

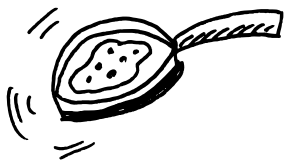
Definition der Funktion
backeEinenPfannkuchen ()

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

Aufrufe anderer Funktionen

Mittels *Funktionen* können Programme in einzelne *Teilbereiche/Teilprobleme* gegliedert werden. 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!

Pfannkuchen



Mal angenommen, wir wollen einen Pfannkuchen backen, dann müssen wir einzelne **Teilprobleme** lösen: Wir müssen 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 können wir den *Pfannkuchen servieren*! Für jedes dieser Teilprobleme definieren wir uns eine eigene Funktion, z.B. die Funktion warteBisBraun ():

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

Aufrufe der Funktion
warteBisBraun ()

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

Definition der Funktion
warteBisBraun ()

Unser Hauptproblem besteht darin, einen Pfannkuchen zu backen. Dies wird von der Funktion **backeEinenPfannkuchen()** erledigt. Die Funktion hat also die Aufgabe, einen Pfannkuchen zu backen und ruft dafür einige andere Funktionen auf, um diese Aufgabe zu lösen. Die Aufgabe wird also in **Teilbereiche** zerlegt, die jeweils von einer eigenen Funktion gelöst werden, z.B. bewirkt der Aufruf der Funktion **warteBisBraun()**, dass unter den Pfannkuchen geguckt wird und solange dieser noch nicht braun ist, wird eine Minute gewartet. Diese Funktion rufen wir zweimal auf, damit der Pfannkuchen von beiden Seiten braun wird! Die Definition der Funktion, also die Festlegung was genau die Funktion machen soll, muss nur einmal implementiert werden. Danach kann man die Funktion beliebig oft im Programm verwenden.

Funktionen mit Parametern

Wenn man einer Funktion beim Aufruf **bestimmte Informationen** übergeben möchte, dann schreibt man in der Definition innerhalb der runden Klammern den oder die gewünschten **Parameter**. Das kann z.B. so aussehen:

Parameter

```
void backeVielePfannkuchen(int anzahl) {
    for (int i = 0; i < anzahl; i++) {
        backeEinenPfannkuchen();
    }
}
```

Eine Funktion kann *einen* oder auch *mehrere* Parameter übergeben bekommen, in unserem Beispiel bekommt die Funktion backeVielePfannkuchen **(int anzahl)** einen Parameter namens **anzahl** vom Typ Integer übergeben. Das bedeutet, dass man beim Aufruf der Funktion eine ganze Zahl mit angeben muss, die bestimmt wie viele Pfannkuchen gebacken werden sollen. Wenn du zum Beispiel alle deine Freunde zum Pfannkuchenessen einladen möchtest:

```
void bereiteAllesVor() {
    aufraeumen();
    saubermachen();
    tueDiesUndDas();
    ...
    backeVielePfannkuchen(200);
    ...
}
```

Durch diesen Aufruf wird jetzt automatisch 200 mal die Funktion backeEinenPfannkuchen() aufgerufen!

Wie oft wird dann insgesamt die Funktion warteBisBraun() aufgerufen? Weißt du's?

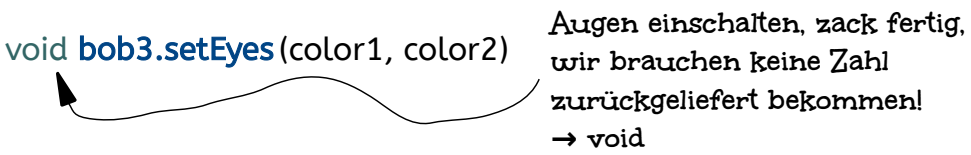
Wenn wir also 200 Pfannkuchen herstellen möchten, dann müssen wir nur noch `backeVielePfannkuchen (200);` schreiben. Wir müssen uns an dieser Stelle nicht um den Teig, das Mixen, das Abwarten usw. kümmern, dies wird alles von unseren anderen Funktionen erledigt!

Funktionen mit Rückgabewert

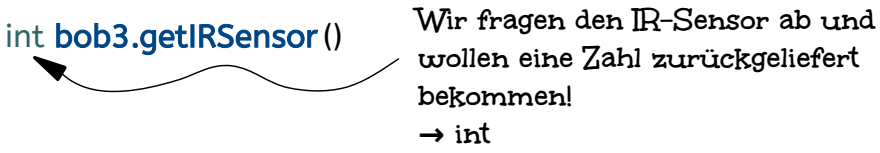
Ein **Rückgabewert** ist der Wert (z.B. eine Zahl), den eine Funktion zurückliefert, wenn sie aufgerufen wird. Manche Funktionen sind **mit** Rückgabewert und manche sind **ohne** Rückgabewert definiert. Das hängt ganz davon ab, was die Funktion machen soll!

Wenn man eine neue Funktion definiert, schreibt man vor den Namen der Funktion ein Schlüsselwort um anzuzeigen, ob die Funktion einen **Rückgabewert** liefert oder nicht:

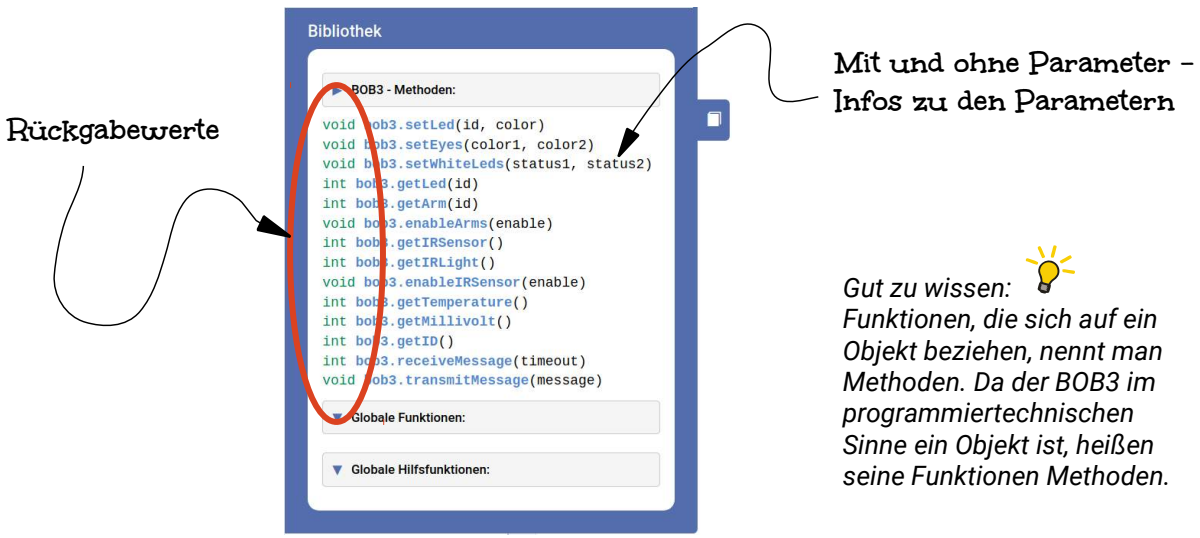
Für Funktionen **ohne Rückgabewert** schreibt man `void`, z.B. hier:



Für Funktionen **mit Rückgabewert** schreibt man den **Typ** des Rückgabewerts, z.B. `int`:



In der Bibliothek von BOB3 kannst du nachschauen, welche vordefinierten Funktionen und Methoden es für den Bob gibt, und wie diese definiert sind:



Gut zu wissen: Funktionen, die sich auf ein Objekt beziehen, nennt man Methoden. Da der BOB3 im programmiertechnischen Sinne ein Objekt ist, heißen seine Funktionen Methoden.

Aufgabe 1: Beschreibe, was eine Funktion ist und welche Vorteile sie bietet.

Aufgabe 2: Beschreibe, was die Funktion `warteBisBraun ()` von Blatt 1 macht.

Aufgabe 3: Beschreibe den Unterschied von Funktionen mit und Funktionen ohne Rückgabewert. Nenne jeweils ein Beispiel!

Aufgabe 4: Welche der folgenden Funktionen/Methoden haben einen Rückgabewert?

a) int `bob3.getIRLight ()`

b) void `bob3.transmitMessage (message)`

c) void `delay (milliseconds)`

d) void `bob3.setLed (id, color)`

e) int `min (zahl1, zahl2)`

f) int `bob3.getArm (id)`

g) int `rgb (red, green, blue)`

h) int `recall ()`

i) void `bob3.setEyes (color1, color2)`

j) void `remember (value)`

Aufgabe 5: Ordne die folgenden Funktionen/Methoden **aufsteigend** anhand der Anzahl ihrer Parameter.

a) int bob3.getIRLight ()

b) void bob3.transmitMessage (message)

c) void delay (milliseconds)

d) void bob3.setLed (id, color)

e) int min (zahl1, zahl2)

f) int bob3.getArm (id)

g) int rgb (red, green, blue)

h) int recall ()

i) void bob3.setEyes (color1, color2)

j) void remember (value)

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 ()

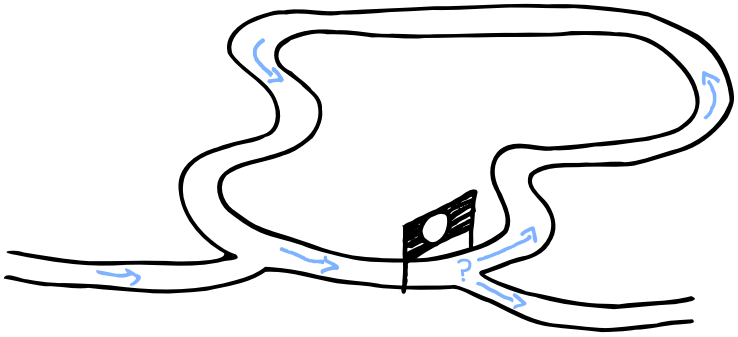
einSchussMineralwasserHinein ()

stelleKeksdoseBereit ()

fuegeMilchHinzu ()

Arbeitsblatt 13 - while-Schleife

Eine while-Schleife dient zur wiederholten Durchführung:

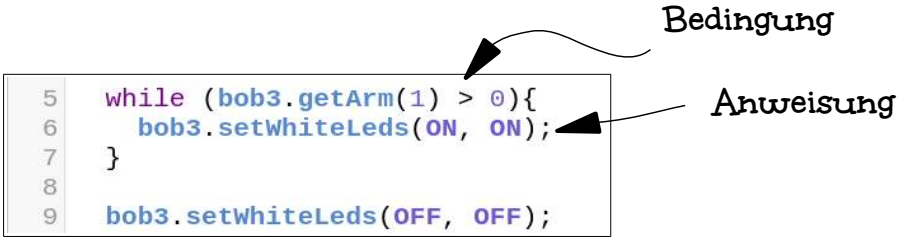


Die Kontrollstruktur **while-Schleife** wird verwendet, um einen bestimmten Programmteil mehrfach zu **wiederholen**. Sie ermöglicht, dass in Abhängigkeit von einer **Bedingung** bestimmte Anweisungen **solange immer wieder** ausgeführt werden, bis die Bedingung **nicht mehr** erfüllt ist.

```
while (Bedingung){
  Anweisungen;
}
```

Solange die Bedingung **wahr** ist, werden die **Anweisungen** ausgeführt. 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.

Wir schauen uns mal ein konkretes Beispiel an:



In **Zeile 5** wird die **Bedingung** `bob3.getArm(1)>0` geprüft: Solange die **Bedingung wahr** ist, also wenn die Abfrage von Arm1 eine Zahl größer als 0 liefert, dann werden beide Bauch-Leds weiß eingeschaltet. Anschließend, wenn der Arm nicht mehr berührt wird, werden die beiden Leds ausgeschaltet. **Solange** wir also den Arm1 anfassen, leuchten die beiden Bauch-Leds!

Was genau ist eine Endlosschleife?

Man kann eine while-Schleife auch als **Endlosschleife** programmieren, wir schreiben dann: `while (true)`. Eine Endlosschleife kann z.B. für eine Alarmanlage sehr nützlich sein:

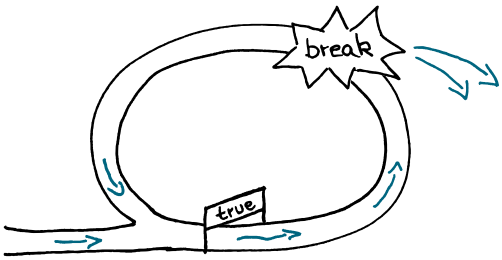
```
4
5   if (bob3.getIRSensor()>8) {
6     while (true) {
7       bob3.setWhiteLeds(ON, ON);
8       delay(200);
9       bob3.setWhiteLeds(OFF, OFF);
10      delay(200);
11    }
12  }
```

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:

Für Schleifen gibt es eine wichtige Anweisung, die **break-Anweisung**. 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**.




```
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      if (bob3.getArm(1) == 3) {
13        break;
14      }
15    }
16  }
```

Abbruch!

Wir verwenden in Zeile 13 eine break-Anweisung, um unsere while-Schleife zu verlassen, also **abzubrechen**, **sobald** Arm 1 unten berührt wird!

Aufgabe 1: Wie oft wird die Schleife `while (100 == 100)` ausgeführt?

- ☐ keinmal
- ☐ einmal
- ☐ zweimal
- ☐ einhundertmal
- ☐ immer und immer wieder

Aufgabe 2: Wie oft wird die Schleife `while (17 < 5)` ausgeführt?

- ☐ keinmal
- ☐ einmal
- ☐ fünfmal
- ☐ siebzehnmal
- ☐ immer und immer wieder

Aufgabe 3: Wie oft wird die Schleife `while (1 != 1)` ausgeführt?

- ☐ keinmal
- ☐ einmal
- ☐ zweimal
- ☐ dreimal
- ☐ immer und immer wieder

Aufgabe 4: Wie oft wird die Schleife `while (1 != 2000)` ausgeführt?

- ☐ keinmal
- ☐ einmal
- ☐ zweimal
- ☐ zweitausendmal
- ☐ immer und immer wieder

Aufgabe 5: Was passiert nach der **break**-Anweisung in Zeile 12?

```
1  #include <BOB3.h>
2
3  void loop() {
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       if (bob3.getArm(1) == 3) {
12         break;
13       }
14     }
15     delay(2000);
16   }
17 }
```

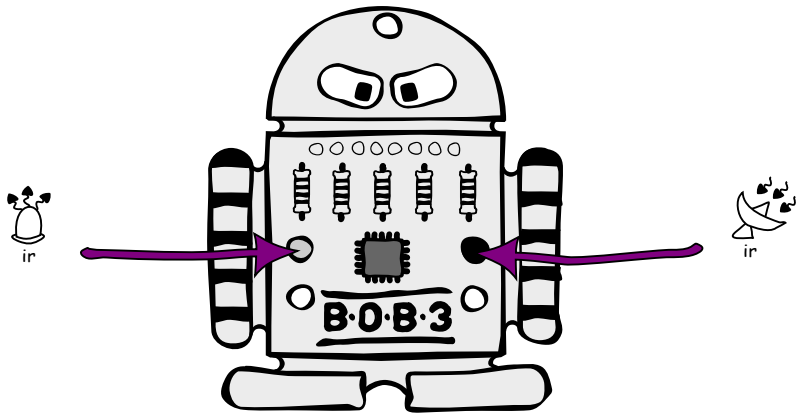
- ☐ Es wird die Zeile 5 ausgeführt
- ☐ Es wird die Zeile 6 ausgeführt
- ☐ Es wird die Zeile 7 ausgeführt
- ☐ Es wird die Zeile 11 ausgeführt
- ☐ Es wird die Zeile 15 ausgeführt

Aufgabe 6: Betrachte das folgende Programm. Für welchen Zweck könnte man es verwenden? Hast du mehrere Ideen? Schreibe deine Ideen auf!

```
1  #include <BOB3.h>
2
3  void loop() {
4
5    while (bob3.getArm(1) == 0) {
6      delay(1);
7    }
8    bob3.setEyes(WHITE, WHITE);
9    bob3.setWhiteLeds(ON, ON);
10   delay(60000);
11   bob3.setEyes(OFF, OFF);
12   bob3.setWhiteLeds(OFF, OFF);
13 }
14
```

Arbeitsblatt 14 - IR-Sensor

BOB3's Infrarotlicht-Sensor



BOB3 hat einen **IR-Sensor**, der aus zwei Teilen besteht: Einer hellblauen **IR-Sende-LED** und einem schwarzen **IR-Empfänger**. Die Abkürzung „IR" steht für „Infrarot". Infrarotlicht ist eine spezielle Lichtart. Die IR-Sende-LED **sendet Infrarotlicht** aus und der IR-Empfänger **detektiert das Infrarotlicht**. Mit diesem Sensor kann der Bob **nah** und **fern** unterscheiden, als Lichtschranke arbeiten, bemerken, ob z.B. deine Hand oder ein Blatt Papier vor ihm ist, oder er kann anderen BOB3-Robotern Botschaften senden!

Für diese Features muss Bob nichts berühren, er bemerkt alles komplett berührungslos mittels des **Reflexions-Verfahrens**. Du kannst z.B. ein Spiel programmieren, bei dem die Reaktionsgeschwindigkeit zweier Spieler gegeneinander antritt: Bob gibt Signale und die Spieler müssen möglichst schnell die **Lichtschranke** auslösen. Am Ende rechnet Bob aus, wer schneller war!

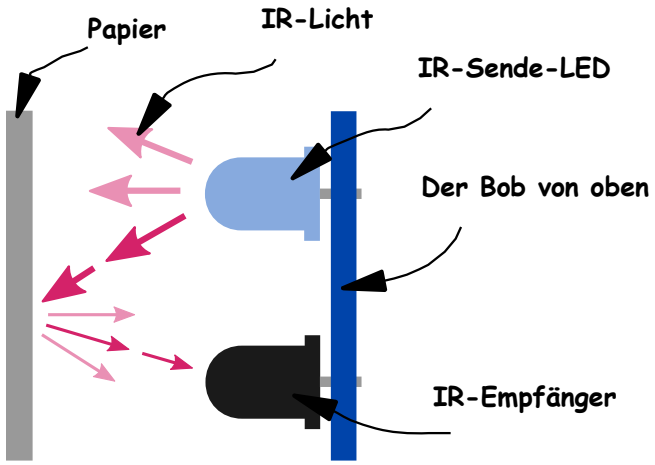
Du kannst Bob auch als **Alarmanlage** programmieren, er kann dann auf Kekse oder andere Dinge aufpassen und schlägt sofort Alarm, sobald er einen Dieb bemerkt!

Reflexions-Verfahren

Die **Detektion** von **nah** oder **fern** funktioniert nach dem **Reflektionsverfahren**:
Die IR-Sende-LED sendet IR-Licht aus, dieses trifft dann auf ein Hindernis (z.B. ein Blatt Papier oder eine Hand), wird von dem Hindernis **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.



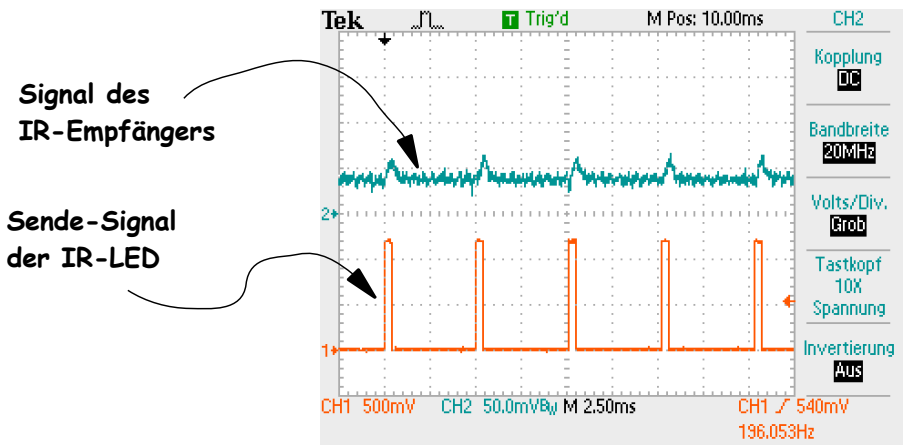
So kann der Bob z.B. auch als Parksensor für eine **Einparkhilfe** programmiert werden:

Fall 1) IR-Sensor detektiert kein Hindernis/anderes Auto
→ LEDs grün, freie Fahrt!
Fall 2) IR-Sensor detektiert ein etwas entferntes Hindernis/anderes Auto
→ LEDs orange, langsamer fahren und aufpassen!
Fall 3) IR-Sensor detektiert ein nahes Hindernis/anderes Auto
→ LEDs rot, STOP!

Messverfahren

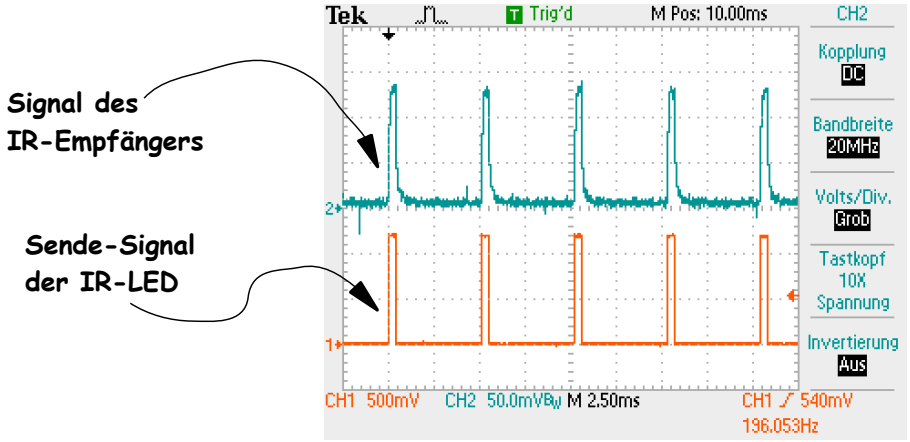
Das Messverfahren lässt sich sehr gut mit einem Oszilloskop veranschaulichen. Hier sieht man als obere Kurve das Signal des IR-Empfängers und als untere Kurve das Sende-Signal der IR-LED.

Beispiel 1: *- Keine Reflexion, kein Tageslicht -*



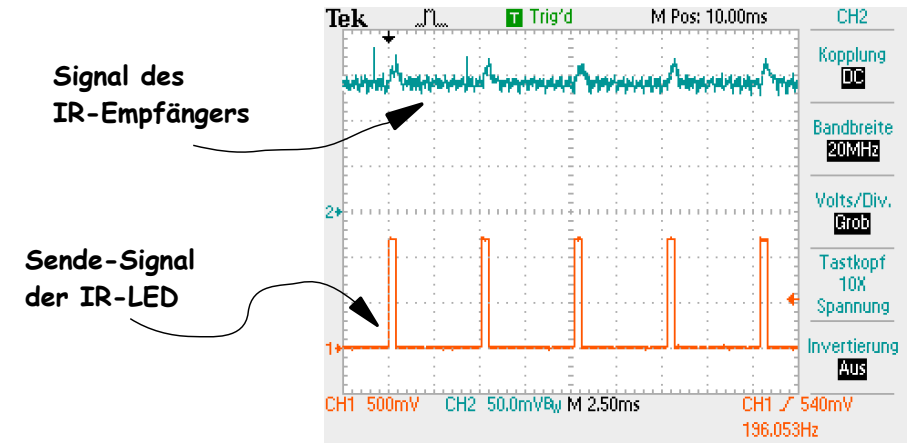
Man sieht an der oberen Kurve deutlich, dass der IR-Empfänger fast keine Signale detektiert.

Beispiel 2: *- Kein Tageslicht, Reflexion an einer Hand -*



In diesem Beispiel zeigt die obere Kurve deutliche Ausschläge sobald die IR-LED sendet. Man erkennt, dass der IR-Empfänger Signale von einem reflektierenden Objekt, in diesem Fall von einer Hand, detektiert.

Beispiel 3: *- Keine Reflexion, Tageslicht -*



In diesem Beispiel zeigt die obere Kurve fast keine Ausschläge, der IR-Empfänger detektiert also kein reflektierendes Objekt. Deutlich zu sehen ist jedoch, dass die Kurve in der y-Achse verschoben ist. An dem höheren Signalpegel erkennt man deutlich, dass der IR-Empfänger in diesem Fall Tageslicht detektiert.

Software-Bibliothek

Zur Ansteuerung des IR-Sensors stehen in der Software-Bibliothek des BOB3 fertig implementierte Methoden zur Verfügung:

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 Methode `bob3.getIRSensor()` liefert den jeweils aktuellen Wert des **IR-Sensors** als Zahlenwert zwischen 0 und 255 zurück. Falls nichts reflektiert wird, ist der Rückgabewert eine 0, falls ein Objekt IR-Licht reflektiert, dann ist der Rückgabewert eine Zahl zwischen 1 und 255, je nach **Abstand** und **Reflexionseigenschaften** des Objekts.

Die Methode `bob3.getIRLight()` kann sehr gut für **Smart-Home** Beispiele verwendet werden, denn sie liefert den aktuellen Wert des **IR-Umgebungslichts** als Zahlenwert zwischen 0 und 255 zurück. Falls der Sensor direktes Sonnenlicht detektiert, so wird ein hoher Zahlenwert zurückgeliefert. Bei absoluter Dunkelheit wird ein sehr kleiner Zahlenwert bzw. eine 0 zurückgeliefert.

Mittels der Methode `bob3.enableIRSensor(enable)` kann der IR-Sensor aktiviert bzw. deaktiviert werden. Nach dem Aufruf von `bob3.enableIRSensor(0)` ist der Sensor dahingehend deaktiviert, das die Reflexionsmessung nicht mehr durchgeführt wird. Die passive Messung des Umgebungslichts findet allerdings weiterhin statt.

Mit der Methode `bob3.receiveMessage(timeout)` und der Methode `bob3.transmitMessage(message)` kann ein 8-Bit-Wert von einem Bob zu einem anderen Bob übertragen werden. Die Methode `bob3.receiveMessage(timeout)` bekommt als Parameter die Anzahl an Millisekunden übergeben, die auf eine Nachricht gewartet werden soll. Falls eine gültige Nachricht empfangen wird, liefert die Methode einen Zahlenwert zwischen 0 und 255. Wird innerhalb des gesetzten Zeitraums keine gültige Nachricht empfangen, dann liefert die Methode eine -1 zurück. So können die Roboter sich untereinander **Botschaften** senden, andere Roboter **fernsteuern** oder sich als **Freunde** oder **Feinde** erkennen!

Aufgabe 1: Beschreibe, aus welchen Teilen der IR-Sensor von BOB3 besteht und welche Funktion diese Bestandteile haben.

Aufgabe 2: Beschreibe, wie das Reflexions-Verfahren funktioniert.

Aufgabe 3: Welche Programm-Codes sind korrekt und würden compilieren? Kreuze die richtigen Antworten an, es sind mehrere Antworten möglich:

- ☐ **bob3.receiveMessage(‘hallo‘);**
- ☐ **bob3.transmitMessage(‘hallo‘);**
- ☐ **bob3.transmitMessage(1000000);**
- ☐ **bob3.transmitMessage(0);**
- ☐ **bob3.transmitMessage(101);**
- ☐ **bob3.transmitMessage();**
- ☐ **bob3.enableIRSensor(enable);**
- ☐ **bob3.enableIRSensor(Enable);**
- ☐ **bob3.getIRSensor(0);**
- ☐ **bob3.getIRLight(100);**
- ☐ **int sensorWert = bob3.getIRSensor();**
- ☐ **int sensorWert = bob3.getIRLight();**

Aufgabe 4: Beschreibe das folgende Programm und nenne eine Anwendungsmöglichkeit.

```
#include <BOB3.h>

void loop() {

  int sensorWert = bob3.getIRSensor();

  if (sensorWert > 10) {
    bob3.setEyes(RED, RED);
  } else {
    bob3.setEyes(GREEN, GREEN);
  }

}
```

Aufgabe 5: Beschreibe das folgende Programm und nenne eine Anwendungsmöglichkeit.

```
#include <BOB3.h>

void loop() {

  int sensorWert = bob3.getIRLight();

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

}
```

Aufgabe 6: Nenne alle zulässigen Parameter der Methode `bob3.enableIRSensor(enable)`.