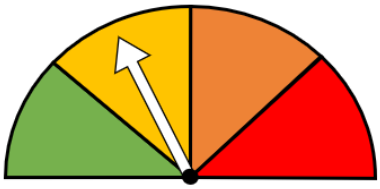
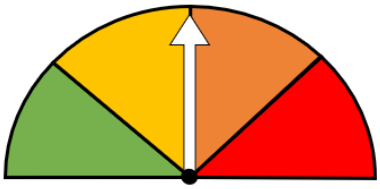
TECHNISCHE UNIVERSITÄT DRESDEN

# Arduino Uno

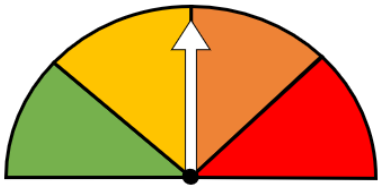
## Station 3 | It‘s the survival of the fittest!



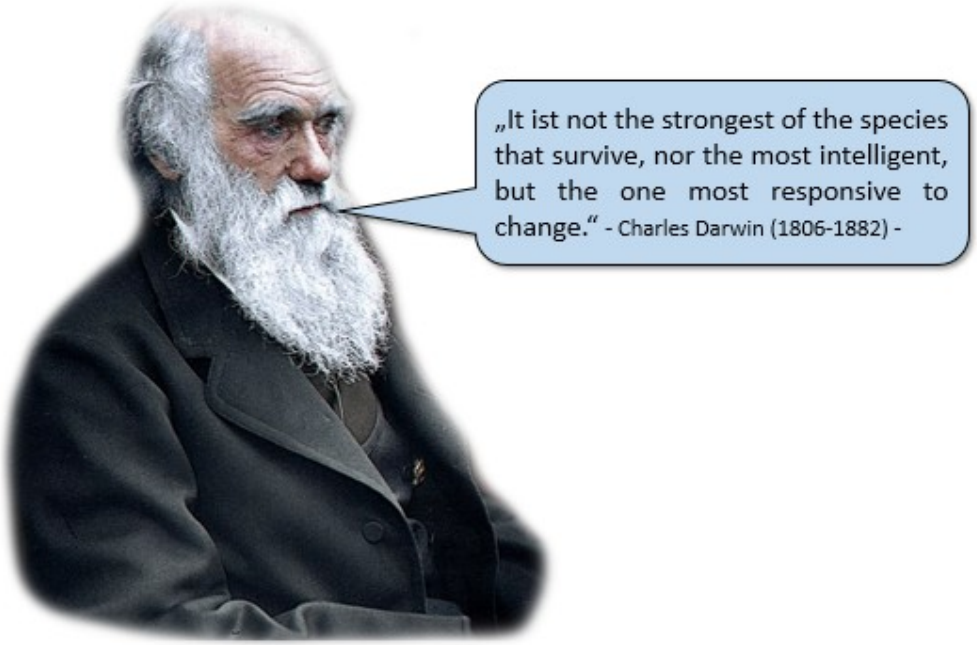
**algorithmisches Denken**



**Programmieraufwand**



**Komplexität der Schaltung**




Professur für Didaktik der Informatik

SYLBER Synergetische Lehrerbildung im exzellenten Rahmen

Das Maßnahmenpaket „TUD-Sylber² – Synergetische Lehrerbildung im exzellenten Rahmen“ wird im Rahmen der gemeinsamen „Qualitätsoffensive Lehrerbildung“ von Bund und Ländern aus Mitteln des Bundesministeriums für Bildung und Forschung gefördert.

GEFÖRDERT VOM Bundesministerium für Bildung und Forschung

## DER NAME IST PROGRAMM – ITS THE SURVIVAL OF THE FITTEST!

Der Name ist Programm! – Eine gute Reaktionszeit kann dir in Gefahrensituationen das Leben retten, zum Beispiel bei einer Notbremsung mit deinem Fahrrad, weil ein Fußgänger dich nicht gesehen hat. Du sollst dich allerdings keineswegs in Gefahr begeben, sondern lediglich ein kleines Spiel entwickeln, um die Reaktionszeit zweier Spielerinnen und Spieler zu vergleichen. Schauen wir mal wer die oder der Schnellste ist!

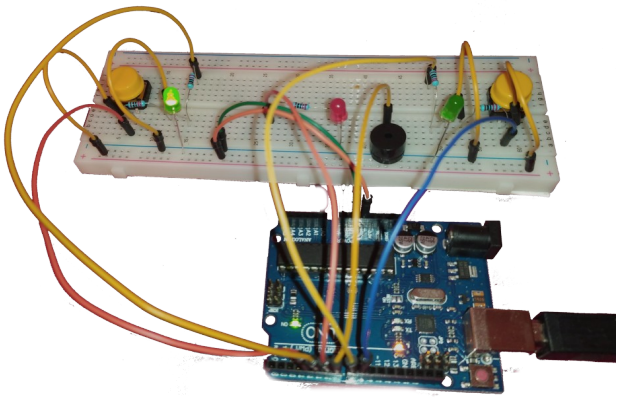
### VORTEST – WIE IST DEINE REAKTIONSZEIT?

Allem voran wollen wir einmal testen wie deine Reaktionszeit ist! Um einen ungefähren Wert zu erhalten kannst du dein Smartphone und die darauf installierte Stoppuhr nutzen. Lass die Stoppuhr laufen bis sie 10 Sekunden anzeigt und drücke dann sofort auf Stopp. Die Differenz zwischen dem angezeigten Wert und den ursprünglich angepeilten 10 Sekunden ist eine Schätzung deiner Reaktionszeit.

Deine Reaktionszeit: _____ Millisekunden

## WIE SOLL DAS PROJEKT AM ENDE AUSSEHEN?

Der Reiz auf den du reagieren sollst wird hierbei eine rote LED sein. Diese soll nach einer zufälligen Zeit ausgehen und dann müsst ihr einen Taster drücken. Der Spieler oder die Spielerin welche/r am schnellsten gedrückt hat wird durch das Aufleuchten einer LED, sowie das Abspielen einer spezifischen Tonfolge über den Piezo-Lautsprecher darüber informiert.



Die Schaltung wird für jeden Spieler / jede Spielerin einen Taster und eine grüne LED beinhalten, sowie eine rote LED und einen Piezo-Lautsprecher. Wie dies am Ende auf dem Breadboard aussehen wird, siehst du in der Abbildung oben rechts. Aber keine Angst! - Auch wenn die Schaltung umfangreich aussieht, werden wir diese Stück für Stück entwickeln.

## MIT ZUFALLSFUNKTION DIE LED STEUERN

Das Ein- und Ausschalten der LED, sowie das Einbringen einer Wartezeit zwischen den Befehlen hast du ja bereits schon einmal realisiert. Dies sah dann wie folgt aus:

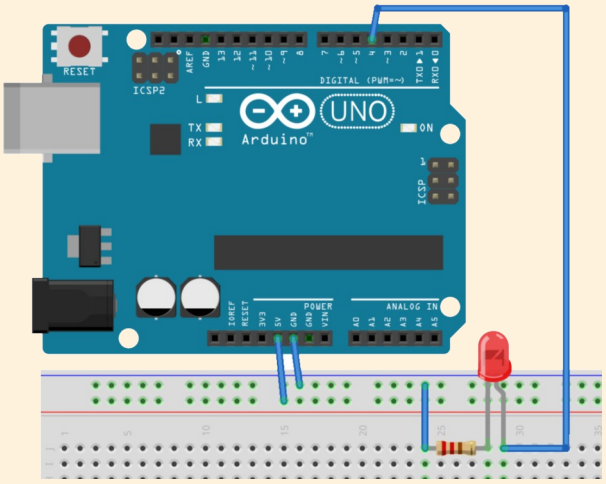
```
int LED = 4;

void setup(){
      pinMode(LED, OUTPUT);
}

void loop(){
      digitalWrite(LED, HIGH);
      delay(500);
      digitalWrite(LED, LOW);
      delay(500);
}
```

### AUFGABE 1 – LED NACH EINER WARTEZEIT AUSSCHALTEN

Als Grundlage für die weitere Betrachtung benötigen wir wieder die bereits bekannte Schaltung mit einer LED. Nutze hierfür eine rote LED, da diese unseren Reiz auf den wir reagieren darstellt. An der Abbildung rechts kannst du dich selbstverständlich orientieren wie diese Schaltung aufzubauen ist, bedenke die Verwendung des Widerstandes!



Im Anschluss schreibst du den notwendigen Programmcode, wie er oben bereits beschrieben ist und testest dieses.

Unter Nutzung der Zufallsfunktion wollen wir nun die LED nach einer zufälligen Zeit ausgehen lassen. Die beiden notwendigen Befehle findest du in der nebenstehenden Tabelle.

| Befehl | Bedeutung |
|---|---|
| `random(200)` | erzeugt eine Zufallszahl zwischen 0 und 200 |
| `random(10, 1000)` | erzeugt eine Zufallszahl zwischen 10 und 1000 |

Wichtig zu wissen ist, dass es sich dabei um **Pseudozufallszahlen** handelt. Dies bedeutet einfach nur, dass wenn der Sketch ausgeführt wird, eine Zahlenfolge erzeugt wird. Aus dieser wählt dann die `random()`-Funktionen einen Zahlenwert aus. Ist es für das Programm von Bedeutung, dass die Zufallszahlen wirklich immer unterschiedlich sein müssen, nutzen wir eine weitere Funktion. Diese legt den Seed auf Basis des Signalrauschens an einem analogen Pin fest. Dabei heißt Seed nichts anderes als Startpunkt.

| Befehl | Bedeutung |
|---|---|
| `randomSeed(analogRead(0))` | zufälliger Startpunkt für den Pseudozufallsgenerator festlegen |

Dementsprechend müssen wir uns nun lediglich eine Variable für die Speicherung der zufällig generierten Wartezeit initialisieren `int wartezeit = 0` und den Wert der Zufallsfunktion in diese Variable speichern über `wartezeit = random(500, 5000)`. Der Seed wird innerhalb der `setup()`- Methode festgelegt, wobei du einen beliebigen analogen Pin für das Auslesen des analogen Wertes nutzen kannst.

## AUFGABE 2 – ZUFALL HINZUFÜGEN

Initialisiere dir zuerst eine Variable für die generierte Zufallszahl. Füge dann innerhalb der `setup()`-Methode die `randomSeed()`-Funktion ein.

Nutze dann die random()-Funktion um die initialisierte Variable mit einem zufälligen Wert zu belegen. In der `delay()`-Funktion darf dann natürlich kein Zahlenwert mehr stehen, sondern die gewählte Variable.

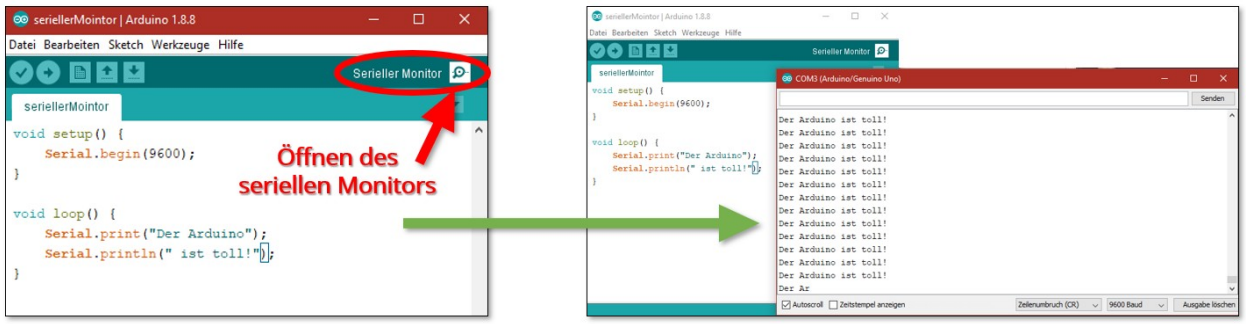
Teste dein Programm mit verschiedenen Grenzen für die Zufallszahlfunktion. Dazu kannst du die Funktion mit einem Zahlenwert nutzen oder auch die mit einer unteren und oberen Grenze.

Warum könnte es hier sinnvoll sein, eine unter Grenze festzulegen?

**Antwort**: _____

Wir wollen natürlich einmal überprüfen, ob die Zahlen sich wirklich voneinander unterscheiden und zufällig sind. Um die Daten an den Computer zu übertragen nutzen wir den sogenannten **seriellen Monitor**, welcher die Informationen die über die USB-Verbindung übertragen werden, als Text darstellen kann. Dazu benötigst du folgende Befehle:

| Befehl | Wo? | Bedeutung |
|---|---|---|
| `Serial.begin(9600)` | `setup()`-Methode | Startet die Verbindung zum seriellen Monitor, wobei die 9600 für die Übertragungsrate steht. |
| `Serial.print("Der Arduino … ");` | `loop()`-Methode | Ausgabe einer Zeichenkette. Ohne Anstriche können auch Variablen ausgegeben werden. |
| `Serial.println(" … ist toll!");` | `loop()`-Methode | Gleiche Funktion wie oben, aber mit einem Zeilenumbruch am Ende. |




Anstelle des Textes fügst du einfach die Variable für die Zufallszahl ein, dabei benötigst du keine Anstriche, sondern nur die Variable in den Klammern.

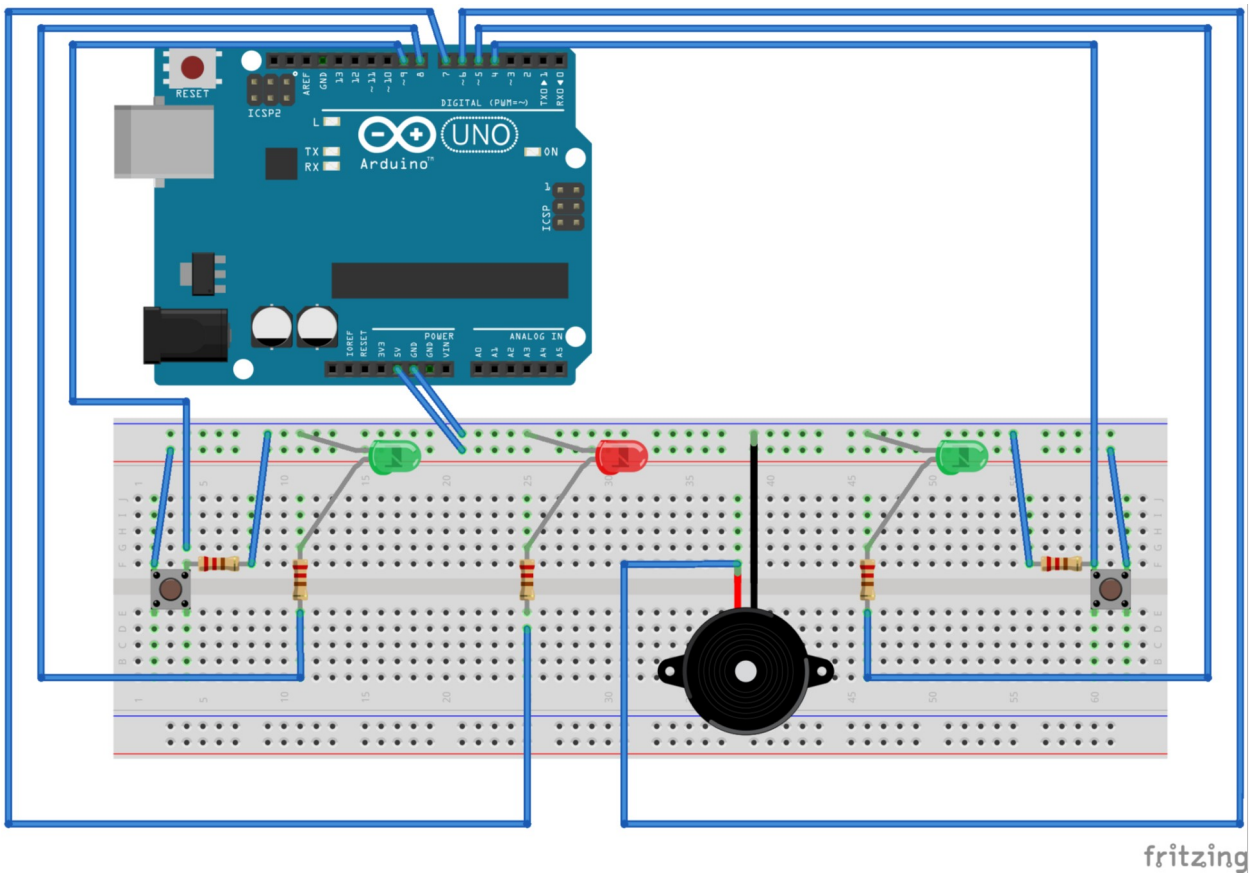
## AUFGABE 3 - AUSGABE DER ZUFALLSZAHL AM COMPUTER

Als erstes startest du den seriellen Monitor mit dem `Serial.begin(9600)` Befehl innerhalb der `setup()`-Methode. Innerhalb der `loop()`-Methode gibst du dann folglich mit `Serial.println(variable)` die aktuelle Zufallszahl aus. Teste auch hier verschiedene Arten der Initialisierung der `random()`-Funktion aus.

## SCHALTUNG FÜR DAS REAKTIONSSPIEL REALISIEREN

Bevor wir uns an die Programmierung heranwagen, realisieren wir die dafür notwendige Schaltung.

Wir brauchen für jeden Spieler / jede Spielerin einen Taster und eine grüne LED, sowie einmal eine rote LED und einen Lautsprecher. Dazu teilen wir das Breadboard quasi in der Mitte in zwei Hälften. Wie genau das aussieht kannst du der nachfolgenden Grafik entnehmen.




### AUFGABE 4 – REALISIERUNG DER SCHALTUNG

Baue die Schaltung auf, wie in der Grafik zu sehen. Da wir sehr viele Bauteile und Kabel haben ist es wichtig dieses übersichtlich zu realisieren. Wähle vor allem auch die richtigen Kabellängen aus, damit die Übersichtlichkeit erhöht wird.

## INITIALISIERUNG DER VARIABLEN

Viele Bauteile bedeuten im Umkehrschluss viele Variablen, deswegen werden wir uns jetzt Schritt für Schritt erstmal mit deren Initialisierung auseinandersetzen.

Die rote LED und eine Variable für die Zufallszahl ist in deinem Programmcode bereits abgehandelt. Weiterführend werden wir für alle übrigen Bauteile die Pinnummer in Integer Variablen speichern.

### AUFGABE 5 – INITIALISIEREN VON VARIABLEN

Nachfolgend sind alle Bauteile aufgelistet, für die du Variablen erstellen musst. Schreibe zur eigenen Kontrolle deinen Code auf die Striche.

Button (Spieler 1): _____

Button (Spieler 2): _____

grüne LED (Spieler 1): _____

grüne LED (Spieler 2): _____

Lautsprecher: _____

## WEITERE PROGRAMMLOGIK

Wir unterscheiden jetzt im Weiteren das Verhalten der roten LED und die Spielereingaben. Die rote LED soll leuchten und nach einer zufällig gewählten Zeit ausgehen. In dieser Zeit soll ein Tasterdruck noch keine Wirkung haben. Innerhalb welcher Methode sollten wir die Steuerung der roten LED platzieren, wenn diese ihr Verhalten nur einmal zeigen soll? - Richtig! In die `setup()`-Methode.

Durch die Nutzung der `delay()`-Funktion wird die `setup()`-Methode angehalten und der Programmcode innerhalb der `loop()`-Methode noch nicht ausgeführt. Somit werden die Spielereingaben in der sich wiederholenden Methode erst überprüft und ausgewertet, wenn die rote LED ausgegangen ist. Die Taster können also im Vorfeld gedrückt werden, ohne den Ausgang des Spiels zu verfälschen.

## DIE SETUP()-METHODE

Das Ausschalten der LED hast du ja bereits schon einmal programmiert, allerdings innerhalb der `loop()`-Methode. Dies kannst du vollständig in die `setup()`-Methode kopieren, da dies ja nur einmal ausgeführt werden soll.

Außerdem müssen wir für die an die Pins angeschlossenen Bauteile noch den `pinMode()` festlegen. Alle drei LEDs legen wir als `OUTPUT` fest und die beiden Taster als INPUT.

### AUFGABE 6 – SETUP()-METHODE VERVOLLSTÄNDIGEN

Vervollständige wie oben bereits beschrieben die `setup()`-Methode. Das heißt einordnen des Programmcodes für die rote LED (mit Ausgabe im seriellen Monitor) und festlegen der Pin Modi.

Teste im Anschluss dein Programm ob die rote LED ausgeht nach zufälligen Zeiten (siehe auch serieller Monitor) ausgeht. Mit Hilfe des Reset-Buttons auf dem Arduino wird das Programm wieder neu ausgeführt.

## WELCHE FÄLLE KÖNNEN WIR UNTERSCHEIDEN?

**Spieler 1 gewinnt**

| Wann? | Was passiert? |
|---|---|
| Button 1 gedrückt | • grüne LED von Spieler 1 leuchtet<br>• Tonfolge 1 abspielen<br>• Ausgabe im seriellen Monitor |

**Spieler 2 gewinnt**

| Wann? | Was passiert? |
|---|---|
| Button 2 gedrückt | • grüne LED von Spieler 2 leuchtet<br>• Tonfolge 2 abspielen<br>• Ausgabe im seriellen Monitor |

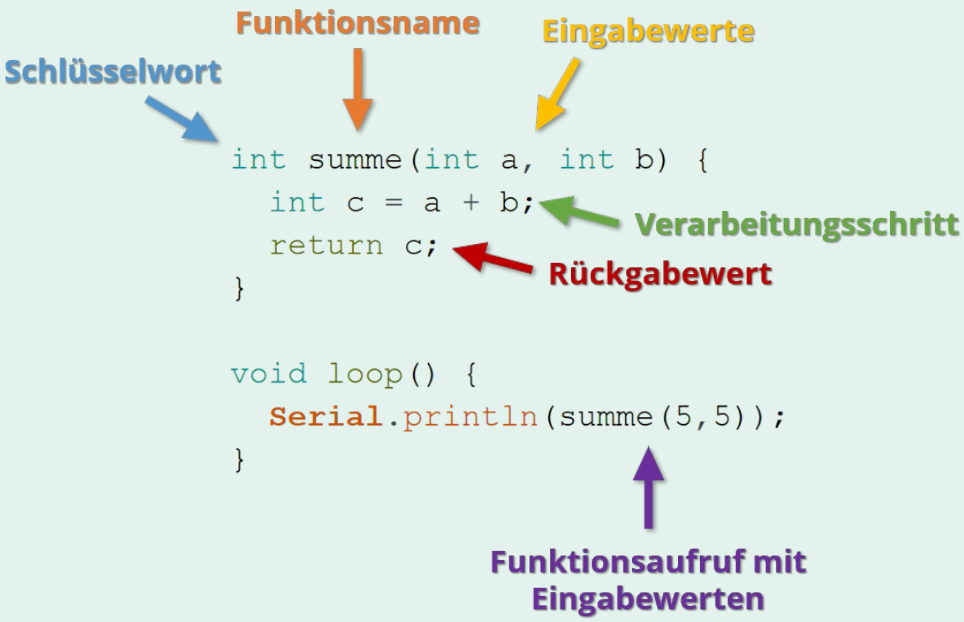
**beide gewinnen**

| Wann? | Was passiert? |
|---|---|
| beide Button gedrückt | • beide grüne LEDs leuchten<br>• Tonfolge 3 abspielen<br>• Ausgabe im seriellen Monitor |

Wir werden uns jetzt genauer die drei verschiedene Fälle anschauen, zu diesem Zweck wollen wir sogenannte **Funktionen und Prozeduren** nutzen.

## WAS IST EIGENTLICH EINE FUNKTION ODER EINE PROZEDUR?

Eine **Funktion** sammelt mehrere Verarbeitungsschritte und erzeugt nach einer Eingabe, eine dementsprechende Ausgabe. Funktionen haben immer einen Namen, mit denen diese aufgerufen werden können. Außerdem können sie einen Eingabe- und einen Rückgabewert enthalten. Das Schlüsselwort für die Funktionsdeklaration beim Arduino ist abhängig vom Rückgabewert. Addieren wir nur ganze Zahlen ist das Schlüsselwort `int`, wollen wir eine Zeichenkette zurückgeben ist dieses `String`. Dieses Wort wird  allem voran gestellt. Ein Beispiel kannst du unten sehen.




Um etwas spezifischer werden, nutzen wir eigentlich nur **Prozeduren.** Der Unterschied liegt darin, das Prozeduren keinen Rückgabewert haben, aber dennoch verschiedene Verarbeitungsschritte ausführen können. Hierbei ist das Schlüsselwort zur Deklaration `void`.  Beispielhaft ist dies am seriellen Monitor unten gezeigt.

```
void ausgabe(){
  Serial.println("Hallo Welt");
}
```

## PROZEDUR FÜR JEDEN MÖGLICHEN FALL

Wir entwickeln nun zusammen eine Prozedur, die anderen gestaltest du anschließend selbstständig! Wir beginnen also zunächst mit dem Schlüsselwort und dem Prozedurnamen:

```
void siegSpielerEins (){ … }
```

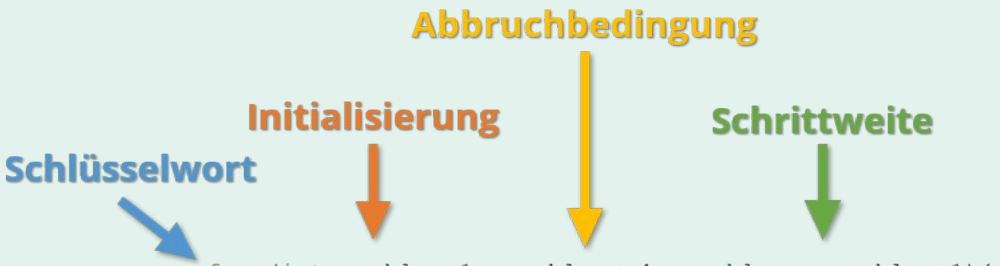
Wenn der Spieler gewonnen hat, soll die grüne LED leuchten, eine Meldung über den seriellen Monitor ausgegeben werden und eine Tonfolge abgespielt werden. Als ersten bauen wir uns das Grundkonstrukt ohne die Tonfolge, die schauen wir uns im Nachgang etwas genauer an.

```
void siegSpielerEins() {
     digitalWrite(LEDspielereins, HIGH);
     >> Tonfolge <<
     Serial.print("Sieg für Spieler 1!")
}
```

Um eine Tonfolge zu wiederholen müssen wir innerhalb der Funktion eine Schleife nutzen. Warum? - Die Funktion wird nach dem Aufrufen nur einmal ausgeführt. Dazu nutzen wir die Zählschleife um zwei Töne vier mal zu wiederholen.

### DIE ZÄHLSCHLEIFE

Die **Zählschleife** wiederholt die in ihr befindlichen Befehle so oft, wie es innerhalb der Schleifendefinition vorgegeben ist. Dazu geben wir einen Startwert (Initialisierung), eine Abbruchbedingung und die Schrittweite vor. Solange diese Bedingung erfüllt ist, werden die Befehle ausgeführt und der Zähler verändert. Beispielhaft sieht das beim Arduino wie folgt aus:



Für unser Projekt wollen wir jetzt jedem Spieler / jeder Spielerin eine spezifische Abfolge von zwei Tönen zuordnen. Für unsere Prozedur ergibt sich dann folgendes:

```
void siegSpielerEins () {
     digitalWrite(LEDspielereins, HIGH);
     for (int zaehler = 1; zaehler<4; zaehler = zaehler+1){
          tone(lautsprecher, 200);
          delay(200);
          tone(lautsprecher, 600);
          delay(200);
     }
     noTone(lautsprecher);
     Serial.print("Sieg für Spieler 1!");
}
```

Unser konstruierte Prozedur musst du zwischen `void loop()` und `void setup()` platzieren. Auf gar keinen Fall darf diese innerhalb einer dieser Methoden stehen.

### AUFGABE 7 - PROZEDUREN VERVOLLSTÄNDIGEN

Übernehme die oben beschriebe Funktion in dein Programmcode. Erzeuge weitere Prozeduren für den Sieg von Spieler/in zwei, sowie für den Fall das beide gewonnen haben. Verändere die Tonfolgen und beachte, welche LED leuchten soll.

## NUR NOCH EIN LETZTER SCHRITT ZUM ZIEL

Sehr gut! Wir haben die Initialisierung, die `setup()`-Methode und die Prozeduren bereits erstellt. Was fehlt nun noch? - Richtig. Wir müssen noch unterscheiden, wann welche Prozedur ausgeführt wird.

Wir müssen es also nun im Programmcode hinbekommen, irgendwie eine Entscheidung zu fällen, welcher Button oder ob beide Button gedrückt wurden. Dazu nutzen wir die sogenannte einfache Verzweigung. Vereinfacht ausgedrückt kann man diese Struktur wie folgt erklären:

**WENN** Button von Spieler eins gedrückt wurde

**DANN** rufe Prozedur siegSpielerEins auf **SONST**  mache nichts

Im Programmcode wird dies dann wie rechtsstehend umgesetzt. Innerhalb der Klammer überprüfen wir unsere Bedingung, sprich ob der Pin des Buttons mit `digitalRead(PIN)` den Wert `HIGH` zurückgibt.

```
if(BEDINGUNG){
      LED _______
} else {
      LED _______
}
```

Um diese Bedingung zu überprüfen nutzen wir den Vergleichsoperator „==". Dieser vergleicht ob beide Werte identisch sind. Außerdem können wir den Zweig mit `else` weglassen, da in diesem Fall nichts ausgeführt wird.

```
if(digitalRead(ButtonSpielerEins) == HIGH) {
      siegSpielerEins();
}
```

Die Unterscheidung für Spieler 1 und Spieler 2 können wir somit problemlos vornehmen. Um die Möglichkeit, dass beide Button gedrückt wurden, benötigen wir eine logische Verknüpfung.

### LOGISCHE UND-VERKNÜPFUNG

Logische Verknüpfen können wir nutzen um mehrere Bedingungen gleichzeitig zu überprüfen. In unserem Beispiel soll die Verzweigung nur ausgeführt werden, wenn Button 1 UND Button 2 gedrückt sind, dass heißt wenn beide Bedingungen wahr sind.

Dazu nutzen wir die UND-Verknüpfung, welche mit zwei kaufmännischen und's „&&" gekennzeichnet wird. Diese gibt nur einen wahren Wert, also true zurück, wenn beide Bedingung erfüllt sind. Veranschaulichen lässt sich dies sehr gut anhand einer Tabelle.

| Bedingung 1 | Bedingung 2 | Rückgabe |
|---|---|---|
| falsch | falsch | falsch |
| wahr | falsch | falsch |
| falsch | wahr | falsch |
| wahr | wahr | **wahr** |

In unserem Programmcode sieht dies dann wie folgt aus:

```
if (buttoneins == HIGH && buttonzwei == HIGH) {
      beidesieg();
}
```

## AUFGABE 8 - PROJEKT VOLLENDEN UND TESTEN

Damit haben wir uns nun auch den letzten Baustein erarbeitet. Was nun noch fehlt, sind die Befehle innerhalb der `loop()`-Methode.

Zunächst liest du den Status der Buttons mit `digitalRead()` aus und speicherst dies in einer neuen Variable, beispielsweise „`int buttoneins`".

Im Anschluss konstruierst du dir mit Hilfe der einfachen Verzweigung die Überprüfungen und rufst die jeweiligen Prozeduren auf.

Damit sollte dein Programm fertig sein. Suche dir einen Mitspieler oder Mitspielerin und testet eure Reaktionszeit!- Viel Spaß dabei!

## DU HAST NOCH NICHT GENUG VON DIESER STATION?

Kein Problem - Frage die Betreuer einfach nach dem Zusatzblatt! Dort wirst du dann noch die Reaktionszeit konkret messen und einen kleinen Fehler beheben, der dir vielleicht beim Testen schon aufgefallen ist.


*Grafik auf dem Deckblatt:* *Charles Darwin, Juliuas Jääskeläinen, CC BY-SA 2.0, https://creativecommons.org/licenses/by/2.0/deed.en, https://commons.wikimedia.org/wiki/File:Charles_Darwin,_English_naturlist,_colored.jpg*
*Screenshots:* *fritzing electronics made by easy und Arduino IDE 1.8.12 (windows)*
*Alle weiteren Grafiken:* *Patrick Binkert, EduInf@TUD*