1. Merkwürdiges Blinken

Lade wieder den originalen Blink-Sketch. Nun soll die LED aber nur genau 5-mal blinken. Dazu verwenden wir zunächst eine if-Abfrage und definieren einen Zähler mit der Zählvariablen `n` durch `int n=0;`

a) Beobachte das Verhalten der LED. Die letzte Anweisung im „if" muss `n=n+1;` lauten.

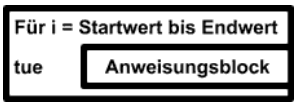
b) Setze nun die Anweisung n=n+1; im Loop **vor** die if-Abfrage. Füge hinter die if-Abfrage (nicht **im** „if") zusätzlich ein Delay von 1ms ein.
Lasse diesen Sketch mindestens 40 Sekunden lang laufen und beobachte die LED. Ist doch merkwürdig oder?

Vielleicht hilft dir folgende Änderung: Ersetze `int n=0;` durch `byte n=0;` später durch `long n=0;` beobachte den Ablauf jeweils genau und vergleiche.

2. Overflow sichtbar machen

Den Overflow aus Aufgabe 1 können wir nun sichtbar machen und in Zeitlupe beobachten. Initialisiere den SM und programmiere eine for-Schleife (s.u.), in welcher eine integer-Variable (i) beginnend von z.B. 32760 hoch-gezählt wird. Setze danach ein delay() auf 1 Sec. damit der Vorgang langsam abläuft. Mit `Serial.println(i);` kannst du den Overflow genau beobachten.

3. Die for-Schleife (Zählschleife)



Mit dieser Schleife kann man fast jeden Anwendungsfall abdecken, bei dem man eine Schleife benötigt aber wie der Name schon sagt, benutzt man diese Hauptsächlich zum Ab - / Durchzählen von abzählbaren Mengen.

Die LED soll wieder 5-mal blinken. Diesmal verwenden wir jedoch eine **for-Schleife**. Die Zählvariable „i" können wir in der for-Schleife deklarieren. Syntax:

```
for(int i=0; i<6; i++) {mach irgendwas;}
```

Allgemein: **for**(Zähler-Initialisierung; Bedingung; Zählerveränderung) **{**Anweisungen**}**.

Möchte man den Wert jeweils um 2 erhöhen, müsste man i=i+2; schreiben. Für´s Runterzählen schreibt man i=i-1; oder kurz i=i-- ;

> ***Info:*** Deklarationen, Initialisierungen und Anweisungen müssen **immer** mit Semikolon (;) abgeschlossen werden. Das was in den geschweiften Klammern steht heißt „Anweisungsblock", danach darf kein Semikolon gesetzt werden.

a) setze die Schleife in das Setup
b) in den Loop

Beobachte jeweils die LED und versuche den Unterschied zu erklären.

4. Die while-Schleife

Bei dieser Schleife wird geprüft, ob die Bedingung zutrifft noch bevor die Schleife betreten wird. Ist dem nicht so, wird die Schleife übersprungen. Eine solche Schleife nennt man „kopfgesteuert". Die while-Schleife wird oft benutzt, wenn noch nicht absehbar ist, wie viele Durchläufe benötigt werden.

Syntax: `while(Bedingungen) {Anweisungen}`,
Beispiel:
`while(sensorLicht < 200){do something;}`. Hier muss die Variable "sensorLicht" vorher deklariert werden.
Oder:

```
int n=0;   while (n<10) {mach etwas; n++;}
```

Realisiere den Blink-Sketch (LED soll 5-mal blinken) mit Hilfe einer while-Schleife. Tipp: Es ist empfehlenswert, den Zähler mit n-- runter zählen zu lassen.
Erweitere den Sketch, so dass nach 5-mal langsam Blinken ein schnelles Dauerblinken entsteht.

5. Plötzlich Töne

a) Öffne wieder den einfachen Blink Sketch und verwende als Ausgang PIN9. Ändere das delay() von 1000ms auf z.B. 2 ms. Schließe nun zwischen PIN9 und GND einen Lautsprecher an. Dann hast du einen Tongenerator. Warum? Erzeuge nacheinander verschiedene Töne.

b) Wir definieren nun einen Zähler, der nach jedem LOOP eins weiter zählt, dann fragen wir in einer `if`-Abfrage ob z.B. n>2000 ist, falls ja erhöhen wir das `delay(zeit);` z.B. von zeit=2 auf 3(ms). Wenn alles stimmt, wird sich die Tonfrequenz nach einer bestimmten Zeit ändern.

6. Bessere Töne: Dreiklang

Mit der Anweisung `tone(9, x, y);` erhältst du eine Tonausgabe, hier an Pin9, Frequenz x, Dauer y. Dazu ist ein Lautsprecher an PIN9 und GND anzuschließen.

a) Erzeuge im Setup einen gut klingenden Dreiklang.

b) Erzeuge im Loop mit Hilfe einer while-Schleife einen Dreiklang, der genau dreimal ertönt.

> ***Info:*** **Subroutine**
> bzw. Unterprogramm, Prozedur, evtl. Methode.
> Subroutinen könnte man als Unterprogramme (Hilfsprogramme) verstehen. Subroutinen sind also keine lauffähigen Programme. Sie liefern nur Hilfsfunktionen, die von einem Programm aufgerufen werden.
> **void** bedeutet „leer", in einer solchen Subroutine wird nichts an das übergeordnete Programm zurückgegeben.