Workshop
Arduino-Programmierung #6

WS2812, blend, Animation "Komet", Gamma-Korrektur

Joachim Baur

E-Mail: post@joachimbaur.de

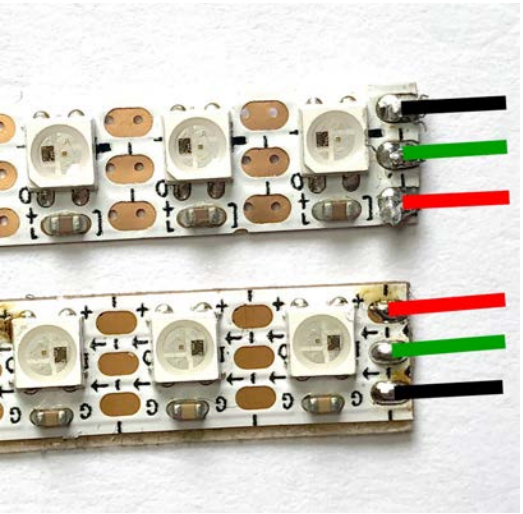
ZTL-Alias: @joachimbaur

Download für diesen Workshop: www.joachimbaur.de/WS6.zip

LED-Streifen schneiden und neu verbinden

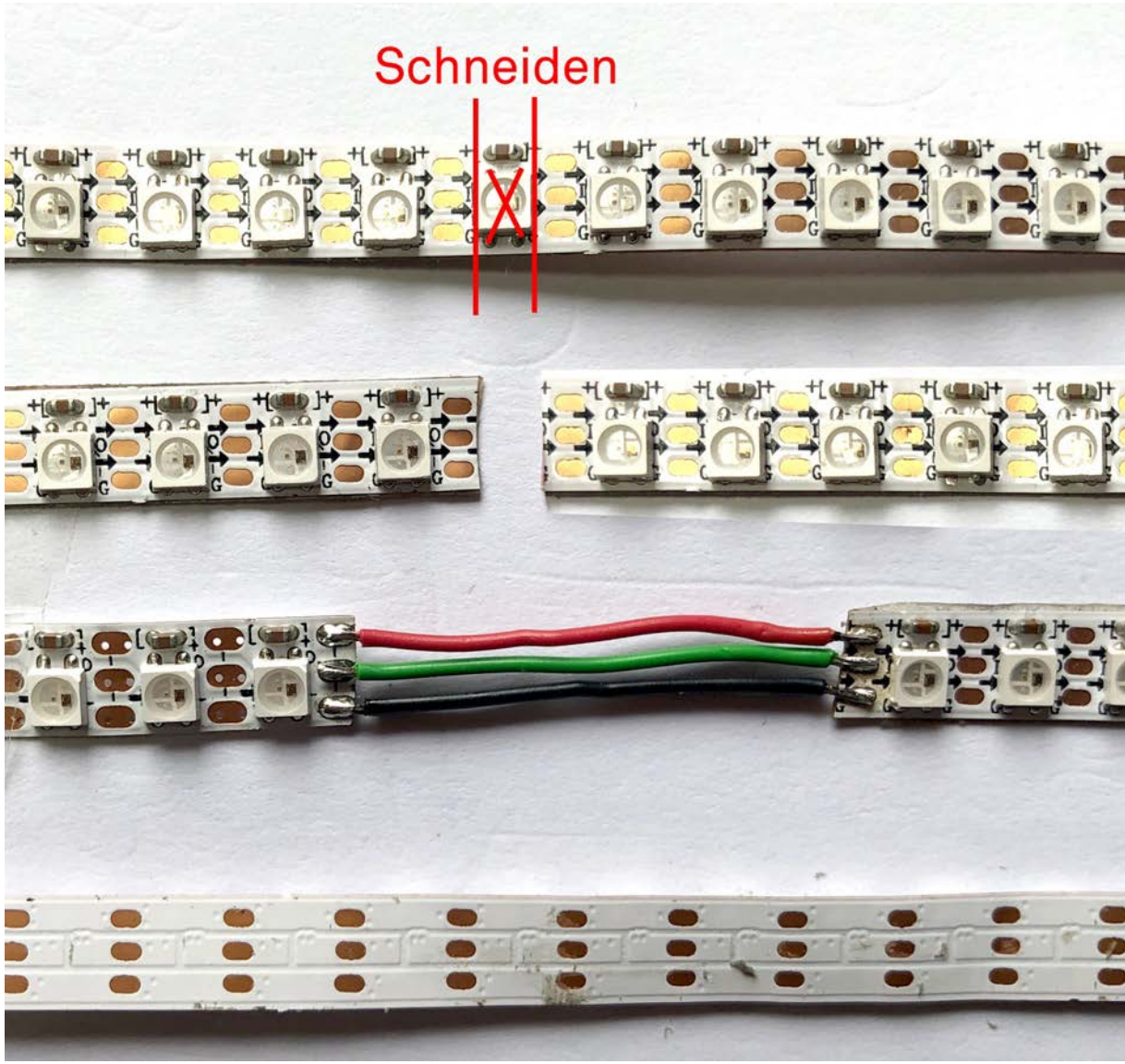
• Streifen immer direkt am LED-Gehäuse schneiden, nicht in der Mitte der Lötpads (1 LED wird geopfert)

• Beim Verlöten zuerst die Lötpads mit Flussmittel verzinnen. Die Kabelenden ebenfalls mit Lötzinn verzinnen. Dann beides zusammen kurz mit dem Lötkolben erhitzen, so dass die LEDs nicht zu heiß werden

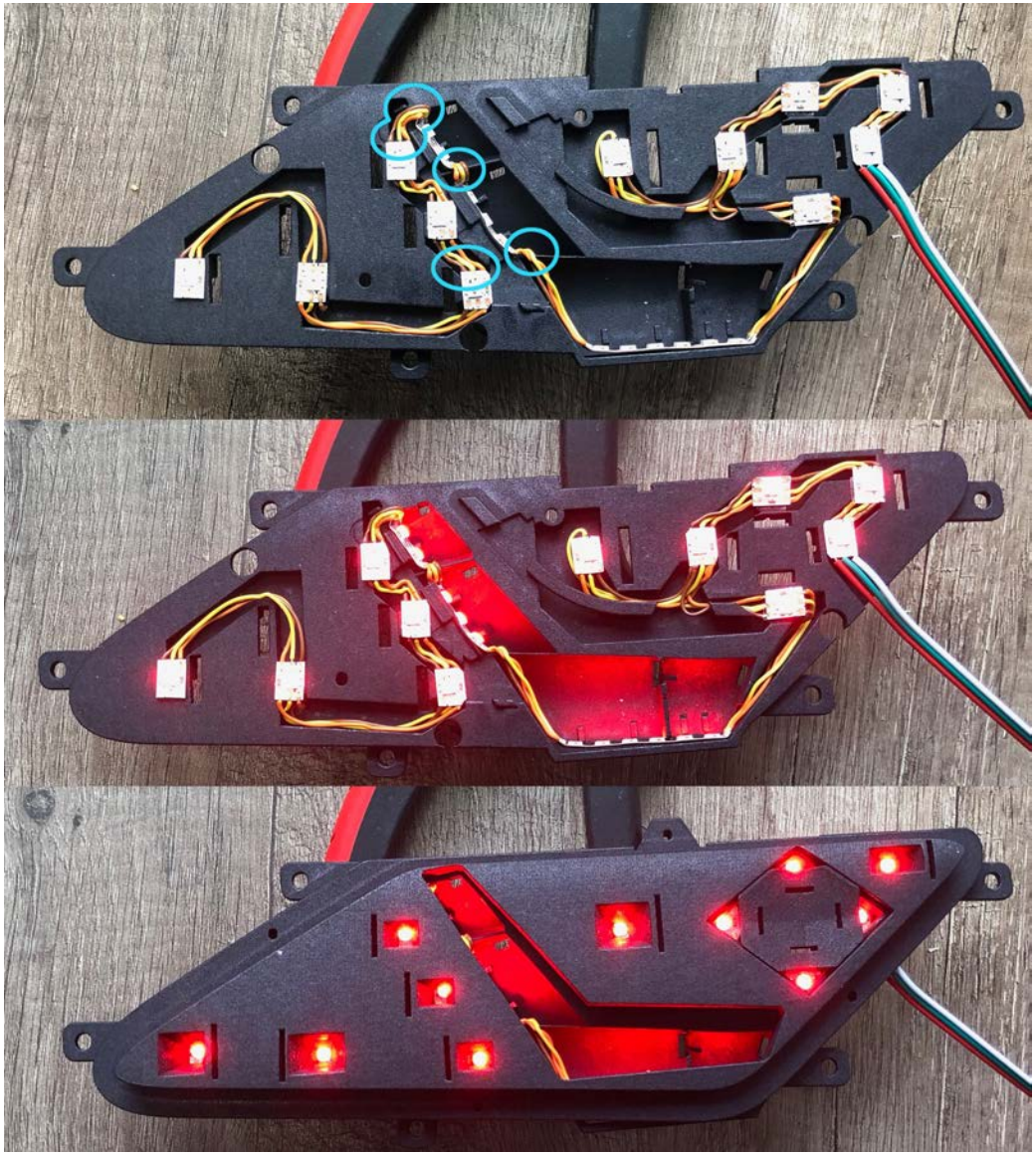


• ACHTUNG: Die Reihenfolge der Lötpads +, data, – kann auch bei gleich aussehenden Streifen unterschiedlich sein!

• Auch die Rückseite der Streifen ist normalerweise mit Lötpads versehen



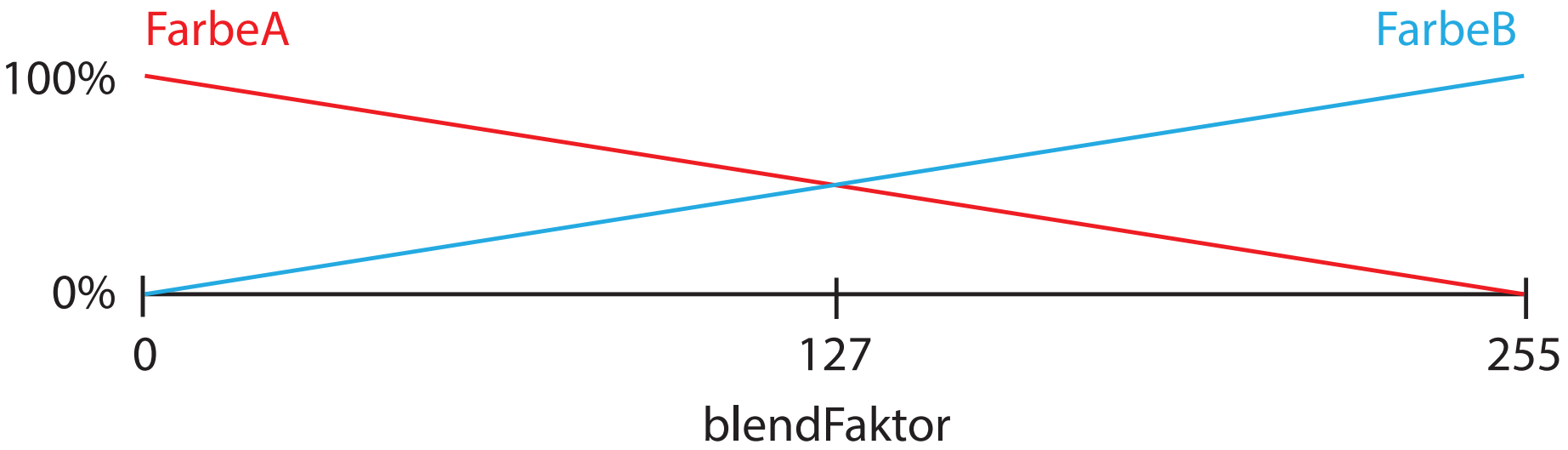
Somit sind beliebige Lichtaufbauten möglich



Farben mischen/überblenden

FastLED-Funktion **blend()** mischt 2 Farben in 256 Stufen

```
CRGB mischFarbe = blend( FarbeA, FarbeB, blendFaktor );
```



blendFaktor = 0: FarbeA 100% FarbeB 0%

blendFaktor = 127: FarbeA 50% FarbeB 50%

blendFaktor = 255: FarbeA 0% FarbeB 100%

Sketch: 23_FastLED_blend.ino

```
#include <FastLED.h>

#define NUM_LEDS 10
#define DATA_PIN 2

CRGB led_array[ NUM_LEDS ];

void setup() {
    FastLED.addLeds<NEOPIXEL, DATA_PIN>(led_array, NUM_LEDS);
    FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);
}

void loop() {
    // von ROT (i=0) zu BLAU (i=255)
    for (int i=0; i<256; i++) {
        CRGB mischFarbe = blend( CRGB::Red, CRGB::Blue, i );
        fill_solid( &led_array[0], NUM_LEDS, mischFarbe );
        FastLED.show();
        delay(10);
    }
    // zurück von BLAU (i=0) zu ROT (i=255)
    for (int i=0; i<256; i++) {
        CRGB mischFarbe = blend( CRGB::Blue, CRGB::Red, i );
        fill_solid( &led_array[0], NUM_LEDS, mischFarbe );
        FastLED.show();
        delay(10);
    }
}
```

Helligkeit beeinflussen

FastLED-Funktion **fadeLightBy()** reduziert die Helligkeit (behält aber die Farbe)

fadeLightBy(n) nimmt Werte von 0…255 für n entgegen

```
led_array[0].fadeLightBy( 127 );
// setzt erste LED auf halbe Helligkeit
```

FastLED-Funktion **setBrightness()** reduziert die generelle Helligkeit für alle LEDs

setBrightness(n) nimmt Werte von 0…255 für n entgegen

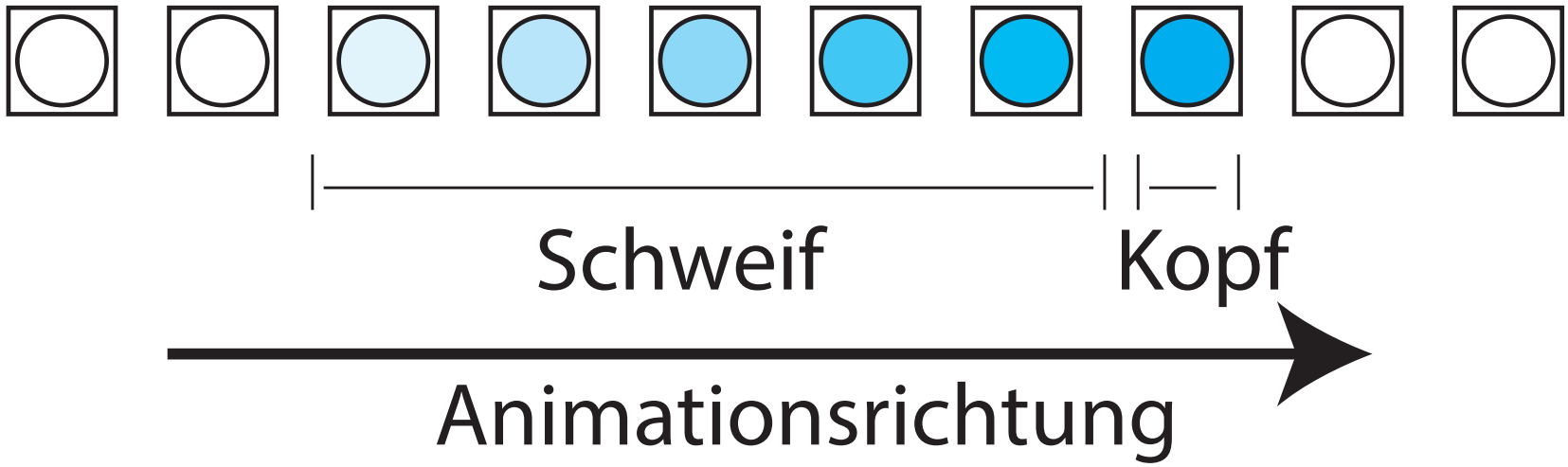
```
void setup() {
    FastLED.addLeds<NEOPIXEL, DATA_PIN>(led_array, NUM_LEDS);
    FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);
    FastLED.setBrightness(100);
}
```

Sketch: 24_FastLED_fadeLightBy.ino

```
void loop() {
    // von BLAU (i=0) zu SCHWARZ (i=255)
    for (int i=0; i<256; i++) {
        // zuerst alle LEDs mit BLAU füllen
        fill_solid( &led_array[0], NUM_LEDS, CRGB::Blue );
        // dann alle LEDs Helligkeit reduzieren
        for (int j=0; j<NUM_LEDS; j++ ) {
            led_array[j].fadeLightBy(i);
        }
        FastLED.show();
        delay(10);
    }

    // zurück von SCHWARZ (i=0) zu BLAU (i=255)
    for (int i=0; i<256; i++) {
        // zuerst alle LEDs mit BLAU füllen
        fill_solid( &led_array[0], NUM_LEDS, CRGB::Blue );
        // dann alle LEDs Helligkeit reduzieren,
        // aber INVERTIERT (255-i)
        for (int j=0; j<NUM_LEDS; j++ ) {
            led_array[j].fadeLightBy( 255-i );
        }
        FastLED.show();
        delay(10);
    }
}
```

Animation „Komet“

- Die Animation soll mit 10 Updates/Sekunde laufen, ohne delay()
- Auf den „Kopf“ des Kometen soll ein 5 LEDs langer „Schweif“ mit abnehmender Helligkeit folgen



- Der Komet soll sich natürlich durch den ganzen LED-Streifen bewegen (vorher alle LEDs schwarz und nachher auch)
- Die Kometenfarbe soll bei jedem Durchlauf eine andere (zufällige) sein

Animation mit 10 FPS (Updates/Sek)

- Wir setzen „Timing“-Variablen ein, um die FastLED-Updates in festen Zeitabständen auszuführen

```
unsigned long naechsteLedUpdateZeit = 0; // in Millisekunden

const int FPS = 10; // 10 Updates (frames) pro Sekunde
const int LED_UPDATE_INTERVAL = 1000/FPS; // in Millisekunden
// LED_UPDATE_INTERVAL = 100 in diesem Fall, da
// 10 Updates pro 1000 Millisekunden = 100 ms je
Animationsphase

void loop() {
    if (millis() > naechsteLedUpdateZeit) {
        // naechsteLedUpdateZeit neu setzen (+100 ms)
        naechsteLedUpdateZeit = millis() + LED_UPDATE_INTERVAL;

        // ...Code für die LED Updates der aktuellen
        // Animationsphase...

        FastLED.show();
    }
}
```

Kometen-Kopf und Schweif erzeugen

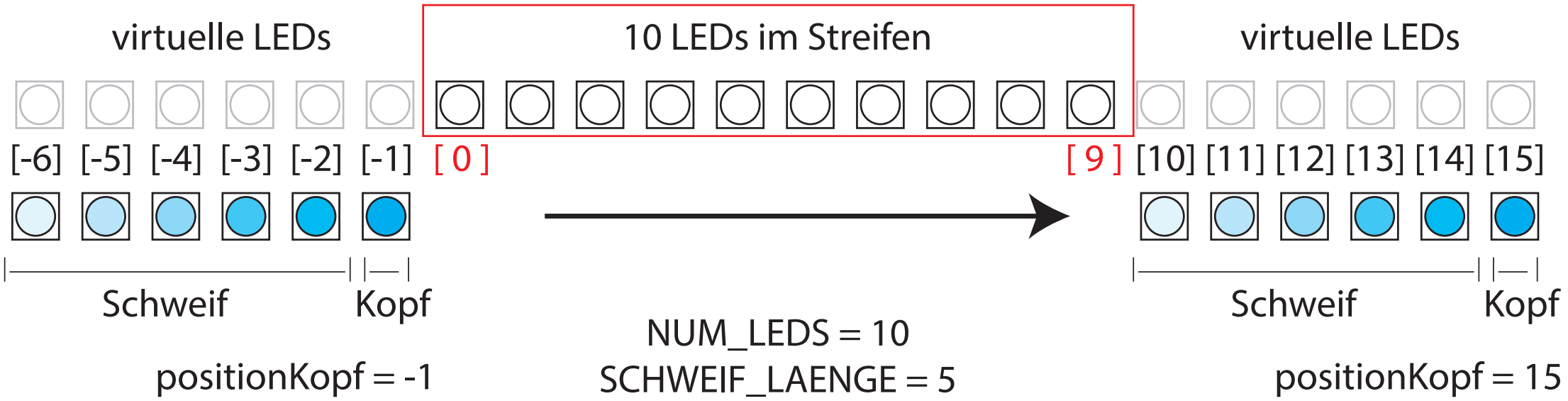
- Zuerst den Kometen statisch an LED-Index 7 (= 8. LED im Streifen!) zeichnen

```
const int LAENGE_SCHWEIF = 5; // Schweif wird 5 LEDs lang
int positionKopf = 7; // Position des Kopfes,
// von -1 (links "vor" dem Streifen)
// bis NUM_LEDS + LAENGE_SCHWEIF (rechts "hinter" dem Streifen)
// erstmal nur fest an Position 7, ohne Animation

CRGB farbeKomet = CRGB::Red; // später Zufallsfarbe

// Im loop() beim LED-Update Code
. . .
    // Kopf zeichnen
    led_array[ positionKopf ] = farbeKomet;

    for ( int i=0; i<LAENGE_SCHWEIF; i++ ) {
        int blendFaktor = map( i, 0, LAENGE_SCHWEIF-1, 10, 250 );
        // map-Zielbereich 10...250: 0 wäre komplett rot,
        // 255 wäre komplett schwarz - beides wollen wir nicht im Schweif
        CRGB mischFarbe = blend( farbeKomet, CRGB::Black, blendFaktor );
        led_array[ positionKopf - i - 1 ] = mischFarbe;
    }
```

Sketch: 25_FastLED_Komet.ino

Komet bewegen

- Damit der Komet vollständig auftaucht und wieder verschwindet, muss die Variable **positionKopf**
- bei -1 starten (“links außerhalb“ des Streifens) und
- bis NUM_LEDS + LAENGE_SCHWEIF (= 15) hochgezählt werden (damit der ganze Komet „rechts außerhalb“ endet



Funktion setzeLedAufFarbe()

- Da die Variable positionKopf jetzt Werte von -1 bis 15 annimmt, das led_array aber nur einen index zwischen 0 und 9 erlaubt, benutzen wir eine Funktion „setzeLedAufFarbe()“ um das led_array zu ändern, anstatt wie bisher direkt mit led_array[index] = farbe

```
void setzeLedAufFarbe( int ledIndex, CRGB ledFarbe ) {
    // zuerst Bereich prüfen!
    if ( ledIndex < 0 ) return; // nicht ausführen, wenn -1 und kleiner
    if ( ledIndex >= NUM_LEDS) return; // nicht ausführen, wenn 10 und größer
    // jetzt ist der ledIndex sicher im Bereich von 0 bis 9 (NUM_LEDS = 10)
    led_array[ ledIndex ] = ledFarbe;
}
```

Sketch: 26_FastLED_Komet_bewegt.ino

```
void loop() {
    if (millis() > naechsteLedUpdateZeit) {
        // naechsteLedUpdateZeit neu setzen (+100 ms)
        naechsteLedUpdateZeit = millis() + LED_UPDATE_INTERVAL;

        // Streifen löschen (voriger Komet wird noch angezeigt)
        fill_solid( &led_array[0], NUM_LEDS, CRGB::Black );

        // Kopf zeichnen
        //led_array[ positionKopf ] = farbeKomet;
        setzeLedAufFarbe( positionKopf, farbeKomet);

        for ( int i=0; i<LAENGE_SCHWEIF; i++ ) {
            int blendFaktor = map( i, 0, LAENGE_SCHWEIF-1, 10, 250 );
            // map-Zielbereich 10...250: 0 wäre komplett rot,
            // 255 wäre komplett schwarz - beides wollen wir nicht im Schweif
            CRGB mischFarbe = blend( farbeKomet, CRGB::Black, blendFaktor );
            //led_array[ positionKopf - i - 1 ] = mischFarbe;
            setzeLedAufFarbe( positionKopf - i - 1, mischFarbe);
        }
        FastLED.show();

        // positionKopf bewegen
        positionKopf++;
        if (positionKopf > NUM_LEDS + LAENGE_SCHWEIF) positionKopf = -1;
    }
}
```

Zufällige Kometenfarbe

- Nach jeder vollständigen Animation soll die Variable **farbeKomet** auf einen zufälligen RGB-Wert gesetzt werden
- Der Farbwechsel wird zum Zeitpunkt des Rücksetzens von positionKopf auf -1 eingefügt:

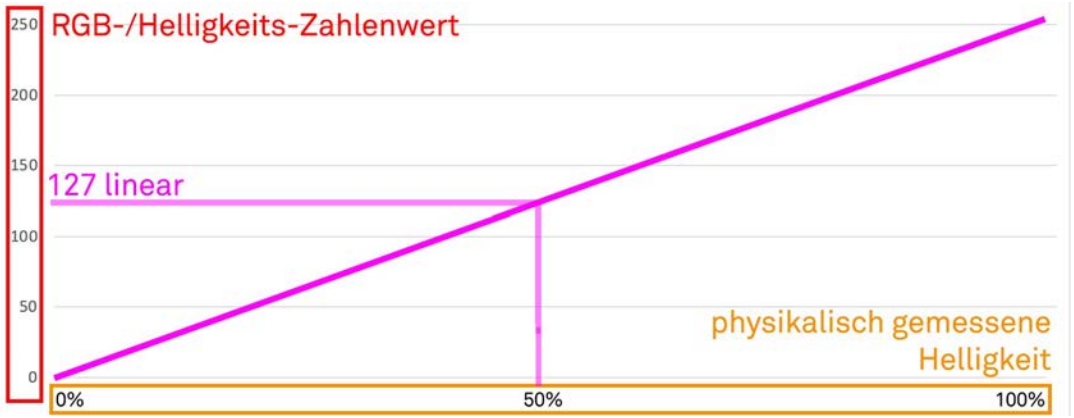
```
randomSeed(analogRead(A0)); // in setup()

if (positionKopf > NUM_LEDS + LAENGE_SCHWEIF) {
    farbeKomet.r = random(256); // Zufalls-Wert von 0 bis 255
    farbeKomet.g = random(256);
    farbeKomet.b = random(256);
    positionKopf = -1;
}
```

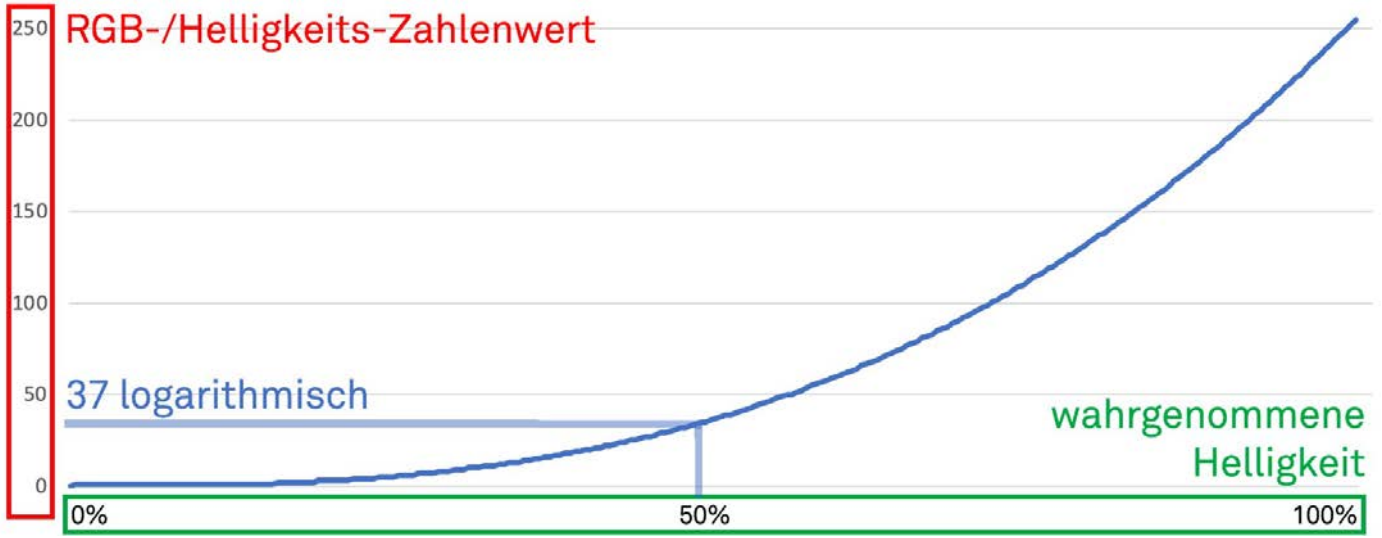
Sketch: 27_FastLED_Komet_farbig.ino

- Aufgabe: Die Kometenfarbe soll zufällig aus einem Array von 5 vorgegebenen Farben gesetzt werden

Gamma

- Der Kometenschweif hat keine gleichmäßig abnehmende Helligkeit, obwohl die map() Funktion gleichmäßig verteilte blendFaktor-Werte erzeugt
- Die Ursache dafür ist, dass die gemessene Helligkeit der LEDs zwar linear ansteigt, die vom Auge wahrgenommene Helligkeit aber logarithmisch:





Gammakorrektur

- Die Anpassung der linearen Werte an diese logarithmische Kurve nennt man Gammakorrektur
- Anstatt den Logarithmus zur Laufzeit im Code zu berechnen (langsam/kompliziert), verwenden wir ein vorberechnetes Werte-Array („Look-Up-Table“):

```
const uint8_t PROGMEM gamma8[] = {
    0, 0, 0, ...
    225,228,231,233,236,239,241,244,247,249,252,255 };
```

- Das Array enthält 256 Gamma-korrigierte Werte für die linearen RGB- bzw. Helligkeitswerte von 0…255

```
wertKorrigiert = pgm_read_byte( &gamma8[ wertLinear ] );
```

https://learn.adafruit.com/led-tricks-gamma-correction/the-issue

PROGMEM und pgm_read_...

- Beim Ausführen von Code werden Variablenwerte und andere Daten im RAM des Arduinos gespeichert
- Da der RAM begrenzt ist, kann ein Teil des Speichers, in dem der Programmcode selbst dauerhaft gespeichert ist, durch den Code als zusätzlicher Speicherbereich benutzt werden
- Dazu wird das Keyword PROGMEM bei der Variablendefinition verwendet. Diese Variablen sind immer const, also zur Laufzeit durch den Code nicht änderbar (read-only)

```
const uint8_t PROGMEM gamma8[] = { . . . };
```

- Um solche Variablenwerte auszulesen, gibt es diverse pgm_read_...-Funktionen, je nach Variablentyp:

```
pgm_read_byte()  -> byte
pgm_read_word()  -> int
pgm_read_dword() -> long
pgm_read_float() -> float
```

```
const uint8_t PROGMEM gamma8[] = {
    0, 0, 0, 0, 0, 0, 0, 0
    228,231,233,236,239,241,244,247,249,252,255 };

. . .

void setzeLedAufFarbe( int ledIndex, CRGB ledFarbe ) {
    if ( ledIndex < 0 ) return
    if ( ledIndex >= NUM_LEDS) return;

    // Gamma-Korrektur auf die r/b/b Werte anwenden
    CRGB farbeKorrigiert;
    farbeKorrigiert.r = pgm_read_byte( &gamma8[ ledFarbe.r ] );
    farbeKorrigiert.g = pgm_read_byte( &gamma8[ ledFarbe.g ] );
    farbeKorrigiert.b = pgm_read_byte( &gamma8[ ledFarbe.b ] );

    //led_array[ ledIndex ] = ledFarbe;
    led_array[ ledIndex ] = farbeKorrigiert;
}
```

Sketch: 28_FastLED_Komet_Gamma.ino

Teiler-Restwert ermitteln („Modulo“)

- In C gibt es den „Modulo“-Operator %
- Er wird analog zu + oder – eingesetzt und gibt den Restwert einer Ganzzahl-Division zurück:

```
int rest = 12 % 3; // rest = 0, da 12/3 = 4
int rest = 10 % 3; // rest = 1, da 10/3 = 3, Rest 1

// wird oft benutzt, um herauszufinden ob eine Zahl gerade
oder ungerade ist:
int ungerade = 7 % 2; // ungerade = 1, da 7/2 = 3, Rest 1
if ( zahl % 2) {
    // zahl ist ungerade...
}
```

- Aufgabe:
  Alle LEDs des Streifens sollen blau leuchten, jede dritte ab der 1. (= LED 1, 4,7 und 10) aber rot
  Der Code soll für beliebige NUM_LEDS funktionieren

Sketch: 29_FastLED_Modulo.ino

```
#include <FastLED.h>
#define NUM_LEDS 10
#define DATA_PIN 2
CRGB led_array[ NUM_LEDS ];

void setup() {
    FastLED.addLeds<NEOPIXEL, DATA_PIN>(led_array, NUM_LEDS);
    FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);
}

void loop() {
    for (int i=0; i<NUM_LEDS; i++) {
        if (i%3 == 0) {
            // index der LED ist OHNE REST durch 3 teilbar
            // also rot
            led_array[ i ] = CRGB::Red;
        } else {
            // es gibt einen Rest beim Teilen durch 3, der
            // Index ist also kein Vielfaches von 3 -> Blau
            led_array[ i ] = CRGB::Blue;
        }
    }
    FastLED.show();
    delay(500);
}
```

Aufgabe: Animation erstellen, so dass die roten LEDs nach außen wandern in einer Endlos-Schleife

Sketch: 30_FastLED_Modulo_Animation.ino

Sketch: 31_FastLED_Cylon.ino

```
#include "FastLED.h"

#define NUM_LEDS 10
#define DATA_PIN 8
CRGB leds[NUM_LEDS];

void setup() {
    FastLED.addLeds<NEOPIXEL,DATA_PIN>(leds, NUM_LEDS);
}

void loop() {
    uint8_t i = beatsin8(20,0,NUM_LEDS-1);
    leds[i] = CRGB(255,0,0);
    FastLED.show();
    fadeToBlackBy(leds,NUM_LEDS,8);
}
```

https://www.reddit.com/r/FastLED/comments/15u2bso/cylon_exampe/

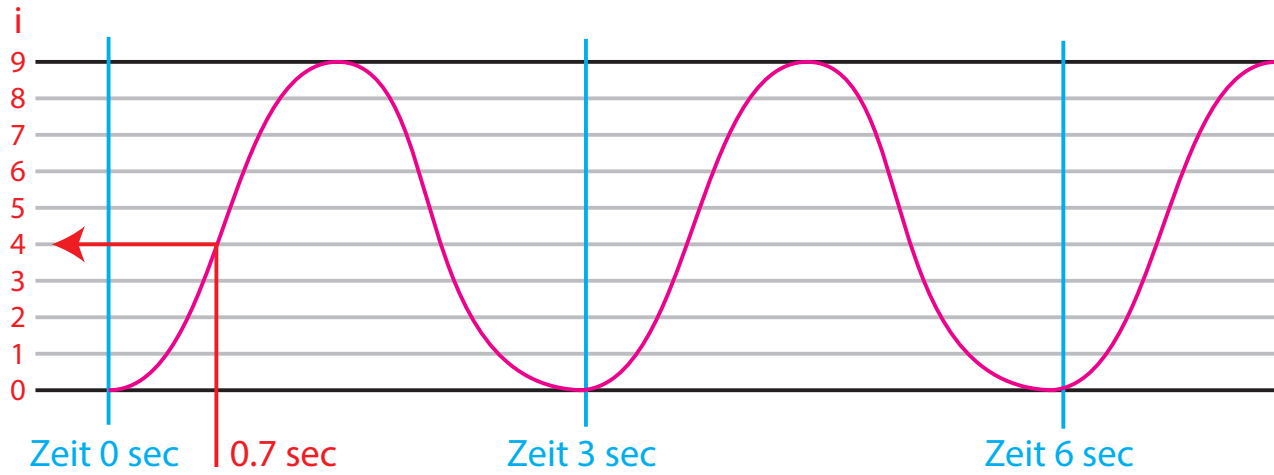
Cylon Funktionsweise

```
uint8_t i = beatsin8(20,0,NUM_LEDS-1);
```

http://fastled.io/docs/group___beat_generators.html#gaecd7cbfc2640407cabf75afcc7d9ddf4

Gibt die Werte einer Sinus-Kurve zurück. Die Kurve hat eine Frequenz von 20 Perioden/Minute (= 1 Periode alle 3 Sekunden) und Amplituden-Werte von 0 bis NUM-LEDS-1

i ist also ein Wert von 0 bis NUM-LEDS-1 und hängt von der Zeit ab, zu der die beatsin8()-Funktion aufgerufen wird




```
leds[i] = CRGB(255,0,0);
```

Die LED am index i wird dann auf voll Rot gesetzt

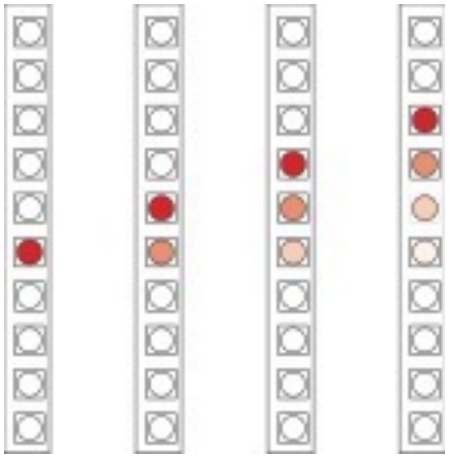
```
FastLED.show();
```

Und der ganze Streifen aktualisiert

```
fadeToBlackBy(leds,NUM_LEDS,8);
```

http://fastled.io/docs/group___color_fades.html#ga404d163d8a422cee2df2baac4151ca98

Danach wird die Helligkeit ALLER LEDs (NUM_LEDS) im Streifen (leds) um den Wert 8 reduziert



```
uint8_t i = beatsin8(20,0,NUM_LEDS-1);
```

Die Schleife beginnt von vorne, nur dass jetzt eine „Spur“ aus alten (= verblassten) roten LEDs hinterlassen wird, Analog zu unserem Kometenschweif