**RainbowWarrior** Bauanleitung



**Benötigte Materialien**

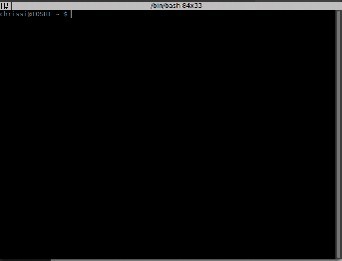
* ESP32
* LED Strip SK6812 RGBW, min. 170mm lang
* Plexiglasplatte - größe nach Belieben. Hier wurde 170 x 170 x 10 mm eingesetzt
* Holzsockel, passend zur Platte. Hier ca 50 x 70 x 190 mm
* Metallknopf als Touch-Sensor (gerne auch aus der Nähkiste, Ersatzknöpfe von Hosen o.ä.)
* Piezo Buzzer (passiv)
* Photoresistor
* Widerstand, 1kΏ (braun – schwarz – rot – gold)
* Lötplatine oder Breadboard
* Jumperwires
* Litzen in 3 Farben (Schwarz, rot, gelb oder andere)
* Lötkolben, Lötzinn, Lötsauglitze
* Holzsäge, Kreissäge oder die Möglickeit, Holz zu verarbeiten
* Bohrmaschine incl. Bohrer in 5mm, ggf. auch 35mm
* Schleifpapier (am besten ein gröberes und ein recht feines)
* ggf. Öl, Lasur, oder Lack um das Holz zu versiegeln

Da sich Python immer weiter verbreitet, habe ich mich dazu entschlossen, das Projekt in MicroPython zu implementieren.   
Dies macht die Einrichtung allerdings etwas komplexer, aber das ist nichts, was man nicht hinkriegen kann :)  
  
Gleichzeitig hat man so die Möglichkeit, mehr auf Kommandozeilenebene zu machen und dort einiges dazuzulernen.

Das Terminal ist sehr mächtig, hier kann man sehr viel kaputt machen, wenn man nicht genau weiß, was man tut.

Aber keine Angst, ich erkläre Dir ganz genau, wie das geht und was hier geschieht!

**Kurze Einführung ins Terminal**



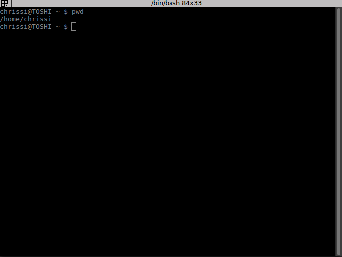
Auf den ersten Blick sieht das Terminal überhaupt nicht spektakulär aus.

Jede Zeile beginnt mit dem Nutzernamen, gefolgt von dem Computernamen (getrennt durch ein @)

Anschließend wird der Ordnername angezeigt, in dem man sich aktuell befindet.

Hier sieht man nur ein unscheinbares „~“, welches aussagt, dass man sich gerade im Home-Verzeichnis befindet.

Abgeschlossen wird die Zeile mit einem „**$**“, das signalisiert, dass man gerade als User im Terminal angemeldet ist.

 **Aktuelles Arbeitsverzeichnis - „pwd“**

Falls Du dir je nicht sicher sein solltest, ob Du Dich im richtigen Ordner befindest, kannst du einfach **„pwd“**  eingeben, was nichts weiter bedeutet als „print working directory“ (was soviel heißt wie „gib akutelles Arbeitsverzechnis aus“)

**Verzeichnis wechseln „cd“**

Um in ein Unterverzeichnis zu wechseln, gibst du einfach den befehl „**cd**“ ein, gefolgt durch den Namen des Unterverzeichnisses. Leerzeichen zwischen diesen beiden nicht vergessen!

Tipp für Schreibfaule: Du kannst auch einfach die ersten Buchstaben des Verzeichnisses eingeben, und mit einem Klick auf die „tab“ taste erledigt die Autovervollständigung den Rest der Schreibarbeit – es sei denn es gibt mehrere Verzichnisse mit dieser Zeichenfolge am Anfang, oder es befindet sich ein Schreibfehler darin.

Möchtest Du in ein ganz anderes Verzeichnis wechseln, gibst du einfach **cd**  ein, gefolgt vom kompletten Verzeichnispfad ein. Dieses Mal muss allerdings ein „**/**“vor den Pfad, um zu signlaisieren, dass es sich nicht um ein Unterverzeichnis handelt

Um wieder ins Übergeordnete Verzeichnis zu wechseln, einfach „**cd ..“** eingeben.

**Dateien kopieren - „cp“**

Um eine Datei zu kopieren, benutzt man den Befehl cp (= copy)

Hierzu wechselst Du mit cd in den Ordner, in dem sich die Datei befindet.

Anschließend gibst du ein: **„cp dateiname.html /home/chrissi/htmlFiles“**

Dieser Befehl besteht aus drei Teilen, getrennt durch ein Leerzeichen:

1. der Befehl cd, gibt an, dass eine Datei kopiert werden soll.

2. die Datei, die kopiert werden solltest

3. der komplette Pfad zum Zielordner.

Wenn du allerdings komplette Ordner kopieren möchtest, brauchst du noch ein **„-r“**, was für Rekursiv steht. So wird der Ordner inklusive allen darin enthaltenen Dateien kopiert.

Das ganze kann dann so aussehen:

**cp -r htmlFiles /home/chrissi**

Ein wichtiger Unterschied zwischen Terminal und der grafischen Benutzeroberfläche: Das Terminal meckert nur, wenn etwas nicht stimmt!

Es ist also völlig normal, dass nach Eingabe des Befehls und Bestätigung mit der Enter-Taste einfach wieder Name/Computername + Arbeitsverzeichnis angezeigt werden! Das signalisiert, dass alles geklappt hat.

Wir verwenden allerdings noch weitere Befehle, die ich an gegebener Stelle nochmals erklären werde.

**Bevor es losgehen kann …**

… sind noch einige Pakete von Nöten, damit wir mit Python arbeiten können, der PC auch eine Verbindung zum Board aufbauen kann und so weiter.

Unter Linux hierzu einfach das Terminal öffnen und folgende Befehle nacheinander eingeben und jeweils mit Enter bestätigen (Achtung: der erste Befehl geht über zwei Zeilen!):

**sudo apt-get install git wget make libncurses-dev flex bison gperf python python-serial python-pip rsync**

**sudo pip install --upgrade pip**

**sudo pip install esptool –upgrade**

Unter Mac OS sind folgende Befehle nötig (ohne Gewähr! Falls etwas fehlen sollte → Google fragen):

**sudo easy\_install pip rsync  
sudo pip install pyserial**

Um das MicroPython Framework auf das Board zu kriegen, hast Du zwei Möglichkeiten.

Entweder Du nutzt das Termial und holst Dir eine Kopie davon auf deinen PC:

**git clone https://github.com/loboris/MicroPython\_ESP32\_psRAM\_LoBo.git**

Oder aber Du gehst auf die Seite „<https://github.com/loboris/MicroPython_ESP32_psRAM_LoBo>“

klickst auf den Button „clone or download“ und lädst dir manuell die Zip-Datei herunter.

Diese musst du allerdings erst noch entpacken.

Anschließend kannst Du entweder mit deinen FileBrowser zuerst in das heruntergeladene Verzeichnis, dann ins Unterverzeichnis „MicroPython\_BUILD“ wechseln, von dort aus mittels Rechtsklick das Terminal öfnen

oder, falls Du auf im Terminal geblieben bist, mit

**cd MicroPythonESP32\_psRAM\_LoBo/MicroPython\_BUILD**

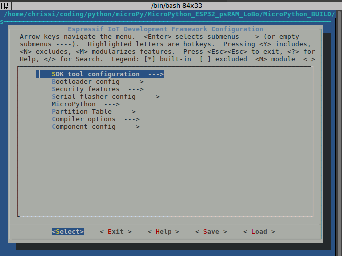
ins entsprechende Unterverzeichnis wechseln.

Jetzt haben wir uns das Framework geholt, anschließend geht’s an die Kernel-Settings.

Eigentlich ist in diesem Schritt nicht viel zu tun, jedoch müssen wir kurz die menuconfig aufrufen, um die sdkconfig zu erstellen.

.**/BUILD.sh menuconfig**

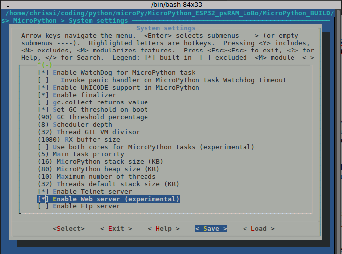
Mit diesem Befehl öffnest du die menuconfig.



Hier navigiert man ausschließlich mit den Cursor-Tasten. Oben/Unten um die verschiedenen Menüpunkte auszuwählen, Rechts/Links um in der Menülieste unten auszuwählen.

Gerne kannst Du dich etwas umschauen und dir Gedanken machen, aber, auch ich weiß bei vielem hier noch nicht, was es zu bedeuten hat, bzw. welche Auswirkungen das hat.

Mit „**select**“ kannst du das Untermenü, bzw. in die Einstellungen des ausgewählten Menüpunktes gelangen. Wenn du einmal nach rechts drückst, sodass „**exit**“ markiert ist, anschließend auf Enter, gelangst Du wieder ins übergeordnete Menü.



Wir müssen hier allerdings die Webserver-Funktion aktivieren.

Dazu wechselst Du in **Micropython → SystemSettings** und navigierst zum Menüpunkt „**Enable WebServer**“. Wenn dieser noch nicht mit einem Stern markiert ist, einmal auf „**y**“ für „**yes**“ drücken um diesen zu aktivieren. Falls Du versehentlich etwas aktiviert haben solltest, das Du nicht wolltest, kannst Du dieses mit „**n**“ für „**no**“ wieder deaktivieren. Zum Beispiel kann man hier auf den Telnet- sowie den FTP-Server verzichten. Es sei denn, Du spielst mit dem Gedanken, einen Fileserver zu erstellen oder etwa Kommandos über das Internet, sprich via WLAN an den Controller zu schicken. Dann könnte Telnet interessant werden.

Das Webserver-Modul benötigt das Modul namens „Websockets“. Dieses befindet sich unter **Micropython → Modules**.

Also mit dem Cursor einmal nach rechts, sodass „Exit“ markiert wird, mit Enter bestätigen, einmal nach unten um auf Modules zu gelangen und ebenso mit Enter bestätigen. Hier einmal „**use Websockets**“ mit y auswählen.

Desweiteren wollen wir den Speicher des Geräts voll ausnutzen. Dazu muss man in **Micropython → System Settings → MicroPython heap size** den Wert **96** eingeben.

Anschließend in der unteren Menüleiste mit Pfeil-Rechts Taste zu „save“ wechseln und mit Enter bestätigen. Du wirst noch einmal gefragt, unter welchem Dateinamen die sdkconfig gespeichert werden soll, hier bitte auch nur mit Enter bestätigen Anschließend 3x „exit“ wählen um die menuconfig zu beenden.

Falls je etwas bei der Ausführung schief gehen sollte oder Du Dir nicht sicher sein solltest, kannst Du ebenso die Datei „sdkconfig“ aus dem Ordner „instruction“ nehmen und in „MicroPython\_ESP32\_psRAM\_LoBoNeu/MicroPython\_BUILD“ einfügen.

Anschließend ebenso die Menuconfig öffnen, mit dem Cursor nach rechts auf „load“, mit Enter bestätigen und die Datei „sdkconfig“ auswählen. Nun – wie oben beschrieben – speichern und beenden.

Jetzt geht‘s an die binary!

Hört sich aber schlimmer an als es ist :)

**./BUILD.sh**

… eingeben … warten … erledigt!

**Endlich Zeit, den ESP auszupacken/anzuschließen!**

Also. Raus aus der Verpackung, ran ans USB-Kabel und einstecken!

Jedes USB-Gerät wird vom PC an einen speziellen Port geknüpft, mit dem es vom System angesprochen wird.

Unter Linux ist dies meist ttyUSB0 oder ttyUSB1.

Sobald der Controller eingesteckt ist, kannst du dies auch einfach prüfen:

**ls dev/ttyUSB\***

**ls** bedeutet list, sprich es listet nun alle angeschlossenen Geräte, die mit ttyUSB anfangen. Der \* signalisiert, dass nach dieser Zeichenfolge noch andere Zeichen folgen

Bevor wir die Binary drauf spielen können, löschen wir vorsichtshalber alles:

**esptool.py --port /dev/ttyUSB0 erase\_flash**

Und spielen dann die binary auf den Controller:

**./BUILD.sh flash**

**Neugierig? Ersten Test??**

Jetzt kommt der entscheidende Vorteil der MicroPython implementierung zum Einsatz!  
Während man bei Arduino-Umgebung oft mühsam durch Recherchen erst herausfinden muss, welche Funktionen zur Verfügung stehen, und erst nach dem Upload auf den Controller herausfindet, ob das so überhaupt funktioniert, kann man das hier mittels REPL (Read Eval Print Loop) ganz schnell.

Also.

Du hast den LED-Strip. Schließe hier das rote und weiße Kabel an eine externe Stromquelle (falls es nur wenige LED sind, kannst du auch den 5V Ausgang und GND vom ESP nutzen. Allerdings kann dies bei Überspannung deinen USB-Port am Rechner zumindest vorübergehend außer Gefecht setzen). Wichtig: um den Stromkreis zu schließen muss der GND der externen Stromquelle mit dem GND des ESP verbunden sein.

Anschließend verbindest Du den **Data-Pin** des LED-Strips (DIN) mit **Pin 14** am ESP.

Öffne den Terminal und gib folgende Befehle ein um dich mit dem Board zu verbinden und REPL zu starten:

rshell --buffer-size=30 -p /dev/ttyUSB0 -a -e nano

repl

Nun wird das benötigte Modul (machine) eingebunden und der Pin für die LED konfiguriert, wobei Du bei pixels=10 anstatt der 10 die Anzahl der LEDs eintragen solltest, die dein Strip hat. Type=1 wenn Du ebenso einen RGBW-Strip hast, ansonsten kann man den type auch auslassen:

import machine

strip = machine.Neopixel(machine.Pin(14),pixels=10, type=1)

Welche Farbe magst du gerne? Cyan, pink? Folgende Farben kann man ganz einfach, ohne Farbwerte zu ermitteln, übergeben:

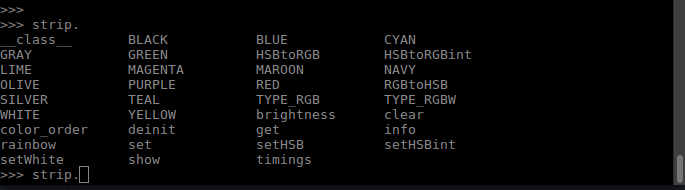
BLACK, WHITE, RED, LIME, BLUE, YELLOW, CYAN, MAGENTA, SILVER, GRAY, MAROON, OLIVE, GREEN, PURPLE, TEAL, NAVY

Mit folgendem Befehl kannst du alle LED auf einmal in der Farbe TEAL leuchten lassen:

strip.set(0, strip.TEAL, num=10)

Du möchtest wissen, welche Funktionen die neopixel library noch mit sich bringt?

Gib einfach „**strip.**“ ein und drücke auf Tab.



**Get the code!**

Nein. Du musst die Software jetzt nicht komplett selbst schreiben!  
Hol Dir einfach, was jemand anderes (in dem Fall ich) schon vorbereitet hat!

Also, zurück im Terminal. Mit Strg+x beendest du REPL, mit Strg+c wird rhsell beendet.

Zuerst solltest Du aber wieder zurück zu deinem Homeverzeichnis wechseln. Dies geschieht am einfachsten, indem man einfach den Befehl „**cd**“ eingibt und diesen mit Enter bestätigt. Anschließend das Git Repository clonen:

**git clone** [**https://github.com/crxcv/Leuchteding.git**](https://github.com/crxcv/Leuchteding.git)

Alternativ kannst Du natürlich immernoch alles manuell vn der Website ([https://github.com/crxcv/Leuchteding](https://github.com/crxcv/Leuchteding.git)) runterladen und entzippen.

**cd Leuchteding**

um in den Ordner zu wechseln.

Und wieder Verbindung zum Board herstellen:

**rshell --buffer-size=30 -p /dev/ttyUSB0 -a -e nano**

Dieses Mal wird aber nicht gleich REPL gestartet, wir müssen ja erst den Code hochladen!

Wenn du gleich den kompletten Ordnerinhalt hochladen möchtest, kannst du das mit **rsync** machen. Dazu musst Du dich in dem Ordner befinden, der hochgeladen werden soll (was Du ja bist, solltest Du der Anleitung gefolgt sein)

Falls das Tool **rsync**  nicht vorhanden sein sollte, kannst Du dies mit „sudo pip3 install rsync“ nachholen

**rsync . /pyboard**

Alternativ kannst Du auch einzelne Skripte hochladen. Hier drei Anwendungsbeispiele:

**cp main.py /flash**

**cp -r www /flash**

**cp www/index.html /flash/www**

Das Erste lädt das Skript namens „**main.py**“ hoch, das zweite den Ordner „**www**“. Das dritte lädt aus dem Unterordner „**www**“ die Datei „**index.html**“ in den Ordner „“**www**“ auf dem Controller.

Aber Achtung: Das dritte Beispiel funktioniert nur, wenn der Ordner www bereits auf dem Board existiert!

Sollte dies nicht der Fall sein kriegst Du aber ne Fehlermeldung.

Einen Ordner kannst du mit dem Befehl „**mkdir**“ (make directory) erstellen:

**mkdir /flash/www**

Softwareseitig sind wir fertig. Zeit zum Basteln!

**Plexiglas**

In meinem Beispiel hatte ich den Vorteil, dass wir in unserer Hochschule einen Lasercutter haben, mit dem man ganz easy Plexiglas schneiden oder gravieren kann.

Da dieser Luxus allerdings nicht vielen zusteht, kannst Du auch auf andere Weise dein Plexiglas bearbeiten. Dort habe ich auch mein Material bezogen.

Allerdings kannst Du auch bei entsprechenden Fachbetrieben nachfragen, ob diese vielleicht brauchbare Reste haben!

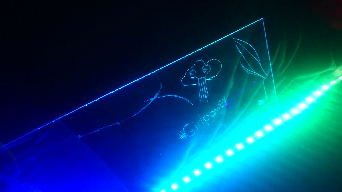
1cm Stärke war von Vorteil, da die LED Strips in etwa diese Breite haben und ich so mit dem Sockel weniger Probleme hatte. Allerdings ist dies kein Muss!

Bevor ich das Plexiglas per Lasercutter geschnitten und graviert habe, hab ich einige Versuche gemacht, wie man dieses alternativ hübsch gestalten kann.

 Am einfachsten ist es, wenn man das Plexiglas von einer Seite großflächig mit feinem Schleifpapier bearbeitet. Dies führt dazu, dass das Licht an der kompletten Oberfläche gebrochen wird und ein schönes diffuses Licht gibt, wie auf dem Bild zu sehen ist.

Optional kannst Du diese Seite auch mit weißer/schwarzer Farbe anmalen/lackieren. Je nachdem welcher Effekt Dir besser gefällt! (Kann einfach getestet werden indem man ein weißes/schwarzes Blatt Papier dahinter legt)

Falls Du einen Dremel oder anderes Werkzeug zur Verfügung hast, kannst Du damit auch hübsche Ornamente, Figuren, Hashtags … reinfräsen.

 Geübte Hände können – allerdings ist hier große Vorsicht geboten – auch mit einem spitzen Gegenstand über die Oberfläche kratzen. Aber die Zeichnungen werden so sehr ungenau, da Rundungen nur sehr schwer machbar sind. Zudem ist die Verletzungsgefahr doch relativ hoch.

**Holzsockel**

Um den Holzsockel zu besorgen, hab ich zuerst einen Baumarkt besucht, um festzustellen: In gesuchtem Format war hier nichts zu finden!  
Vielleicht hast auch Du Glück und findest ein Stück Holz in passender Größe zum Plexiglas in der Nachbarschaft!  
Ich bin daher mit meinem Fahrrad einmal durchs Industriegebiet und hab bei einer Firma, die Holzreste vor der Werkstatt liegen hatte, einfach mal nett nachgefragt. Die waren sogar sehr froh, dass ich ihren „Müll“ abgenommen habe!   
  
Oder vielleicht hat deine Familie Holzreste in passender Größe!

Meine Variante ist bewusst einfach gehalten. Der Holzsockel ist ca 55 x 70 x 190 mm groß. Groß genug um die Scheibe mit 10 x 170 x 170 mm zu tragen.

Um die Plexiglasscheibe im Sockel zu befestigen war ein Schlitz mit 1cm breite nötig.

Dazu kannst Du auch deinen Vater oder andere handwerklich begabte Menschen fragen, ob die dies für Dich übernehmen können, falls Du dich nicht rantraust. Wir haben hierzu eine Handkreissäge eingesetzt, die ehrlich gesagt auch mir zu schwer war.

Um eine schöne Oberfläche herzustellen kannst Du nun den Sockel mit Schleifpapier großflächig abschleifen. So wird die Oberfläche sehr fein. Hier wirst Du wahrscheinlich schnell feststellen, dass das Holz sehr empfindlich auf Anfassen reagiert – man sieht dies sehr schnell. Daher sollte man das Holz nun mit Öl, Lack oder Lasur versiegel und anschließend gut trocknen. Dafür sollte man allerdings – abhängig vom eingesetzten Material, mindestens 24 Std. einplanen.

 Anschließend müssen die Löcher für die Kabel gebohrt werden.

Ich habe mich für Bohrer mit einer Stärke von 5mm entschlossen. So hatte ich keine Probleme, bis zu 5 Kabel durch ein einziges Loch zu kriegen.

Eines für den Touch-Sensor wurde vorne mittig platziert, zwei weitere (für die LED, den Piezo und den Photoresistor) befinden sich mit ca. 1cm Abstand vom Rand jeweils seitlich im Spalt, der für die Plexiglasplatte vorgesehen ist.

Falls dies möglich ist, kannst Du mit größeren Bohrern (35mm) von unten oder hinten Aussparungen für den Controller in den Sockel bohren.

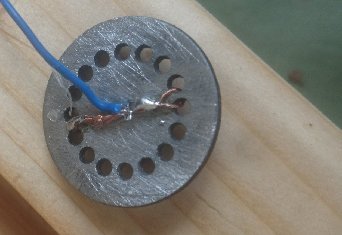
Anschließend sollte der

**Jetzt geht‘s ans Zusammenbasteln!**

 Die LEDs müssen zuerst mit langen Kabeln verlötet werden. Je nach Format sollte die Länge angepasst werden. Ich habe jeweils ca. 20-25 cm Kabel genutzt.

Ebenso sollte der Piezo und der Lichtsensor mit langen Kabeln versehen werden.

Den Metallknopf an einer Seite mit Schleifpapier gut anrauhen, damit die Leitfähigkeit nicht durch Schmutz beeinträchtigt wird.

 Mein Metallknopf hat praktischerweise sehr viele Löcher, die sich als Ösen nutzen lassen. Hier habe ich vorsichtshalber mal zwei Schlaufen verlötet. Dieser wird einfach mit der Heißklebepistole auf dem dafür vorgesehenen Bohrloch befestigt. Achtung! Nicht die Finger verbrennen! Es wird doch sehr heiß, wenn man den Knopf, und auch gleich die LED an den noch heißen Klebstoff presst.

Die LEDs befestigst Du ebenso mit Heißklebepistole am Plexiglas.

Bevor das Plexiglas in den Holzsockel gesteckt werden kann, müssen die Kabel für Lichtsensor und Piezo-Buzzer durch die entsprechenden Löcher geführt werden. Anschließend können auch die Kabel fürs Plexiglas durchgeführt werden.

**Nun stecken wir alles zusammen!**

Leider müssen hierbei einige Jumperwires („male“) dran glauben, um die Kabel mit dem Board zu verbinden. Wenn Du diese in der Mitte durchschneidest, kannst Du gleich zwei Kabel mit den Board verbinden.

Kabel kann ma einfach zusammenlöten, indem man die Enden abisoliert und diese erst einmal mit Lötzinn verzinnt. Anschließend hält man beide Enden zusammen und verlötet diese.

Falls vorhanden, stülpe zuerst ein Stück Schrumpfschlauch über eines der Kabel. Damit lassen sich ungewollte Fehlkontakte und auch Kurzschlüsse verhindern. Diesen anschließend mit einem Feuerzeug vorsichtig erhitzen, damit er sich zuammenziehen kann.

Alternativ würde auch Isolierband gehen, aber bei Kabel in vorhandenem Durchmesser ist es sehr schwer, dieses anzubringen.

Falls Du nicht löten möchtest, kannst Du auch vorhandene male-female Wires und ein Breadboard nutzen. Die Länge dürfte gerade so ausreichen.

Folgendermaßen werden die Bauteile verbunden:

- **Touch-Sensor** mit GPIO02 auf dem Board

- (+), bzw. rotes Kabel vom **Piezo Speaker** mit GPIO22

- (-), bzw. schwarzes Kabel vom **Piezo Speaker** mit GND auf dem Breadboard

- (+), bzw rot vom **LED Strip** an 5v VIN am Board, besser noch an eine externe Stromquelle

- (-), bzw schwarz vom **LED Strip** an GND auf dem Breadboard (wenn externe Stromquelle genutzt wird muss GND vom Breadboard auch mit dem GND der externen Stromquelle verbunden werden

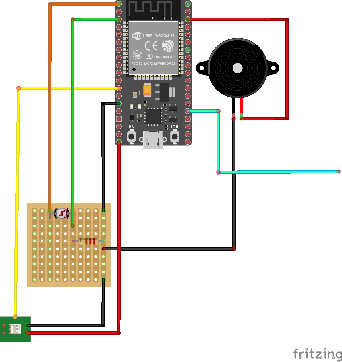
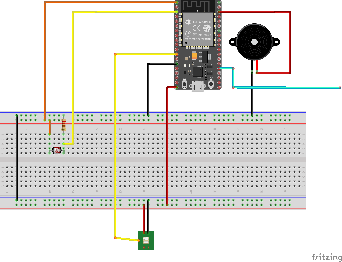
- DIN, oder gelbes Kabel vom **LED Strip** an GPIO14 auf dem Board

- eine Seite des **Photoresistors** an 3.3V auf dem Board

- die andere Seite des **Photoresistors** wird mit einem Widerstand dazwischen mit GND verbunden

- zwischen **Photoresistor** und Widerstand werden die Werte ausgelesen. Also muss hier eine Verbindung zu GPIO36, bzw SVP hergestellt werden.

Da dies sehr verwirrend sein kann, habe ich zwei Schaltpläne erstellt. Links mit Breadboard, rechts mit Lötplatine. Das cyanfarbene Kabel soll den Touchsensor symbolisieren!



**Neugierig?**

**Dann wage einen Start!**

Hier benötigst Du kein Terminal mehr.

Zuerst musst Du dich mit deinem Handy/Pc per WLAN mit dem Board verbinden.

Suche unter den verfügbaren Netzwerken einfach nach „**ESP32**“ und gib als Passwort „**HuchEinPw**“ ein.

Falls dir ein anderer Netzwerkname lieber sein sollte, öffne einfach die Datei „connectSTA\_AP.py“ mit einem Editor Deiner Wahl und ändere die Zeilen 9 und 10 nach Belieben:

ssid = "esp32"

pw = "HuchEinPw"

Hier hast Du auch die Möglichkeit, die Verbindungsdaten zu Deinem Heimnetzwerk einzutragen!

Hierzu musst Du Zeile 5 und 6 ändern:

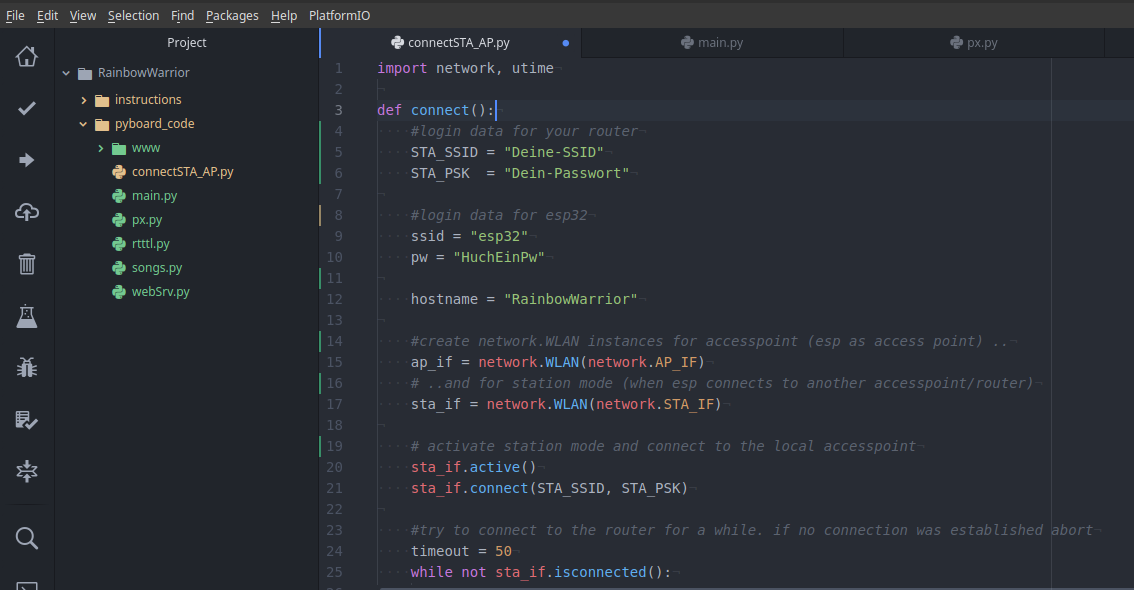
STA\_SSID = "deineSSID"

STA\_PSK = "deinPasswort"

So musst Du Dich nicht jedes Mal mit dem Netzwerk des Controllers verbinden, wenn Du diesen bedienen möchtest.

Selbstverständlich kannst Du einfach einen vorhandenen Editor nutzen. Aber ich zeig Dir gerne mal, was an den Alternativen (die auch Profis nutzen) so schön ist.

Ich nutze für Projekte dieser Größenordnung gerne AtomIO. Kostet nichts, läuft auf allen Betriebssystemen und ist beliebig erweiterbar. Hier kann man (wenn man es kann) eigentlich alles anpassen. Man muss nur wissen, wie :)



Dir fällt sicherlich auf, wie schön bunt das ganze aussieht!  
Dasist keinesfalls reine Dekoration – das nennt sich „Syntax-Highlighting“.   
Das Wort „Syntax“ hast Du vielleicht schon im Deutschunterricht gehört. Ist auch ein Teilgebiet der Grammatik :) und hat – wie bei der Sprache eben auch – mit dem sinnvollen Aufbau des Textes, oder hier, des Codes, zu tun.

Durch die farbliche Markierung wird alles wunderschön übersichtlich!   
Zudem hat man auf der linken Seite gleich die komplette Ordnerstruktur des Projekts im Überblick und kann auch per Rechtsklick weitere Projektordner hinzufügen.

Die Webseite erreichst Du ganz einfach unter der Adresse **rainbowWarrior.local.**

Einfach in der Adresszeile im Browser eingeben!

Aber auch diesen Namen kannst Du Dir nach Belieben anpassen!   
Schau einfach auf **Zeile 12** auf der Datei **connectSTA\_AP.py** und passe den Namen nach Deinem Belieben an!   
Allzu lange Namen sollten nicht verwendet werden. Ist auch sehr umständlich, das einzugeben :)

Du musst nur darauf achten, dass Du den Namen mit Ausführungszeihen versiehst!  
So weiß das Programm, dass es sich hierbei um einen sogenannten String, also eine Zeichenkette handelt.

Nichts weiter als verschiedene Zeichen auf der Tastatur in einer Reihe. Gegebenenfalls getrennt durch Leerzeichen, Sonderzeichen und so weiter.

Kleine Einschränkung gibt es allerdings doch! Backslashes („\“) sollte man nicht verwenden!   
Die werden nämlich in Verbindung mit bestimmten Buchstaben eingesetzt, um Dinge wie einen Zeilenumbruch („\n“) oder eine Einrückung mit der Tab-Taste („\“) mit wenigen Zeichen darzustellen. Möchtest Du wirklich einmal in einem String (außerhalb der URL versteht sich) einen Backslash einsetzen, musst du diesen doppelt tippen („\\“)!

Aber nun zurück zu unseren Spielereien.

Auf der Seite **rainbowWarrior.local** kannst Du auswählen, welches Lichtmuster angezeigt werden soll.   
Leider ist diese URL nur mit dem PC erreichbar. Leider kann ich es Dir nicht erklären warum. Aber falls es auch bei Dir auf dem Handy nicht möglich sein sollte, musst Du die Seite mit der IP Adresse „**192.168.4.1**“ aufrufen.

Unter **rainbowWarrior.local/alarm**, bzw „**192.168.4.1/alarm**“ kannst Du die Systemzeit einstellen, sowie die Weckzeit auswählen. Ebenso hast Du hier die Möglichkeit, den Wecksound abzuspielen. Hierzu habe ich einige Melodien gefunden!

Möchtest Du alle LED in einer Farbe Deiner Wahl leuchten lassen, besuche die Seite **rainbowWarrior.local/led**.

Hier musst Du nur auf eine Farbe deiner Wahl klicken/tippen, schon leuchten alle LED in dieser Farbe!

Den Touchsensor kannst Du im Normalbetrieb nutzen, um durch die verschiedenen LED-Animationen zu wechseln. Dabei wird mit einer Berührung die Animation gestartet, mit der nächsten wieder beendet, dann erst die darauf folgende aufgerufen.

Wenn Du über den Webserver ein Lied oder eine Animation gestartet hast, werden diese durch Berührung ebenso wieder deaktiviert.

Und wenn Du Dir einen Wecker gestellt hast, wird auch dieser durch eine Berührung des Sensors wieder deaktiviert.

**Viel Spaß damit!**

**Du möchtest auch mal eine eigene Animation schreiben?**

Dann legen wir mal los:

Als erstes musst Du eine neue Datei erstellen. Nennen wir sie mal „**animation.py**“.

Anschließend müssen wir überlegen, welche Module für das, was wir tun wollen, benötigt werden.

Hier möchte ich mit Euch eine einfache, aber doch schöne Animation kreieren, die sich „Sparkle“ nennt. Bei dieser Animation wird immer nur eine LED an einer unbestimmten Stelle für eine bestimmte Zeit leuchten, bevor sie wieder ausgemacht wird und die nächste dran ist.

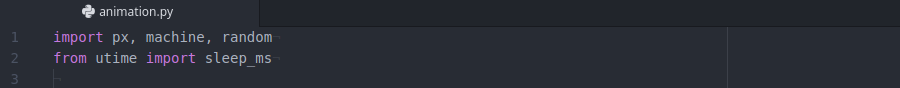
Auch die Farbe wird zufällig bestimmt.

Alleine, um die LEDs anzusteuern brauchen wir die Funktionen für den Pin und für die Neopixel. Beide befinden sich im Modul „machine“.

Da die LED-Position und die Farbe zufällig bestimmt werden, brauchen wir den Zufallsgenerator. Englisch „random“. Also müssen wir das Modul „random“ einbinden.

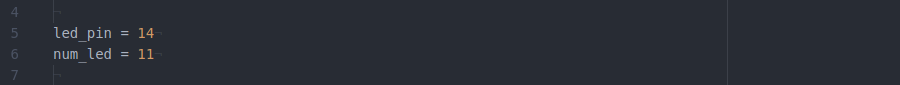
Zusätzlich müssen wir eine bestimmte Zeit warten, weshalb aus dem Modul „utime“ die Funktion „sleep\_ms“ eingebunden werden muss, die bewirkt, dass für eine bestimmte Anzahl von Millisekunden die Ausführung des Programms an dieser Stelle aussetzt.

So sieht das ganze dann aus:



Im nächsten Schritt definieren wir, an welchem Pin der LED Strip angeschlossen wird und wieviele LEDs an diesem vorhanden sind.

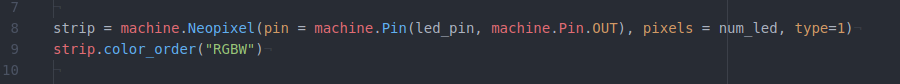
Diesen Schritt nennt man auch „Variablendeklaration“ und „Wertzuweisung“.



Anschließend legen wir eine Instanz der Klasse „Neopixel“ an.

Um dies zu tun sind bestimmte Angaben wichtig. Zuerst muss der Pin übergeben werden.

Dieser Pin muss aber auch dem „machine“ Modul bekannt sein. Nemen der Nummer des Pins muss auch bekannt sein, ob an diesem Pin Signale eingehen, also zum Beispiel von einem Sensor ausgelesen werden, oder ob es sich um ausgehende Signale handelt. Wir wollen hier dem Strip mitteilen, welche LEDs er schalten soll, daher muss dieser Pin als ausgehend deklariert werden.



„pin = machine.Pin(14, machine.Pin.OUT)“ erledigt das ganze.

Außerdem muss mitgeteilt werden, wieviele LEDs an dem Strip vorhanden sind und um welchen Typ es sich handelt. Hast Du einen Strip mit LEDs in drei Farben (rot, gelb, blau), ist der Typ 0 zu verwenden. Ich habe jedoch LEDs mit vier Farben (rot, gelb, blau, weiß), daher muss ich der Typ 1 verwenden.

Die „color\_order“ gibt an, in welcher Reihenfolge die Farben übergeben werden. Wenn ich zum Beispiel sagen möchte, der Strip soll rot leuchten, würde ich nur die Werte 255 für rot und jeweils die 0 für grün und blau mitgeben.   
Wenn nun die !color\_order“ nicht „RGB“ sondern „GRB“ ist (was bei Typ 0 häufig der Fall ist), würden die LED grün leuchten, anstatt rot.   
Beim RGBW Strip ist die Reihenfolge standardgemäß aber RGBW. Trotzdem musste ich diesen Schritt der Vollständigkeit halber einfügen.

Um die Animation aus dem Terminal zum Beispiel aufrufen zu können, müssen wir sie in eine sogenannte Funktion packen. Nennen wir sie mal „sparkle“. Wie wir diese aufrufen können erkläre ich später.

Damit der Code nicht nach einmaliger Ausführung, sprich nach einmaligem Aufleuchten einer LED stoppt, muss hier eine Schleife eingebaut werden.  
Ich habe mich für eine Endlosschleife entschieden.



Statt dessen kannst Du aber auch sagen, dass die Schleife nur eine bestimmte Zeit laufen soll.

Dies würde man mit einer sogenannten „for-schleife“ erledigen.

for i in range(2000):

Damit würde dieser Code genau 2000 Mal ausgeführt werden.

Wichtig ist hierbei, dass jede Zeile Code, die zur Funktion gehört, 1x mit Tab eingerückt wird. So wird signalisiert, was denn un in der Funktion ist und was nicht. Ebenso muss mit jeder Zeile verfahren werden, die wiederholt ausgeführt werden soll. Diese Zeilen sind dann im Vergleich zu den Deklarationen 2x mit Tab eingerückt.

Wie bereits erwähnt werden die LED, die leuchten soll, sowie die Farbe durch Zufall bestimmt. Dies erreichen wir mit random. random.randint(0, 10) gibt zum Beispiel eine zufällige, ganze Zahl zwischen 0 und 10 zurück.

Der seltsame Wert, der bei color genutzt wird, erklärt sich dadurch, dass man die Farbe nicht nur als einen Wert für rot, einen für blau und einen für grün übergeben kann. Farben werden auch gerne als Hexadezimalwert angegeben. Der höchste Wert, der so möglich ist, ist also „FFFFFF“ (wobei die ersten beiden für den Wert für rot, die zweiten beiden für grün und die dritten für grün stehen). Dieser Wert konvertiert in einen dezimalen Wert ergibt 16777215



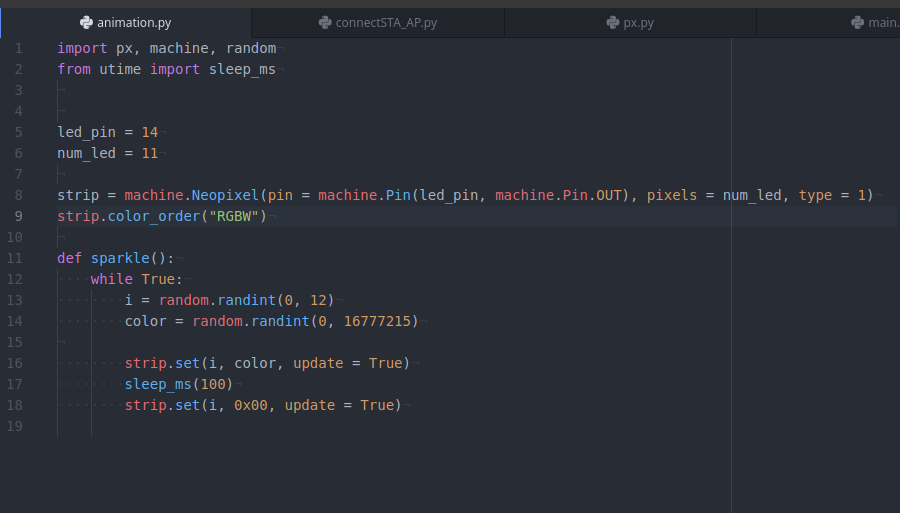
Im letzten Schritt bringen wir die LED zum leuchten!

Strip.set() braucht mindestens zwei Werte: einmal die LED, die geschalten werden soll und den Farbwert. update=True ist optional. Man kann hier auch update = False einsetzen. Das wäre nötig, wenn man zum Beispiel im selben Durchlauf nicht alle auf die selbe Farbe setzen möchte. Wenn aber update mit False angegeben ist, muss anschließend strip.show() augeführt werden, ansonsten gehen die LED nicht an!



sleep\_ms(100) bedeutet, wie bereits erwähnt, dass die Ausführung des Codes für 100 Millisekunden ausgesetzt weden soll. Falls dir diese Animation zu schnell sein sollte, kannst Du gerne mit den Werten etwas herumspielen.

So sieht das ganze dann komplett aus:

****

**Animation ins Programm einfügen**

Dazu benötigen wir drei Dateien:

**px.py**, hier sind alle LED Animationen gespeichert,

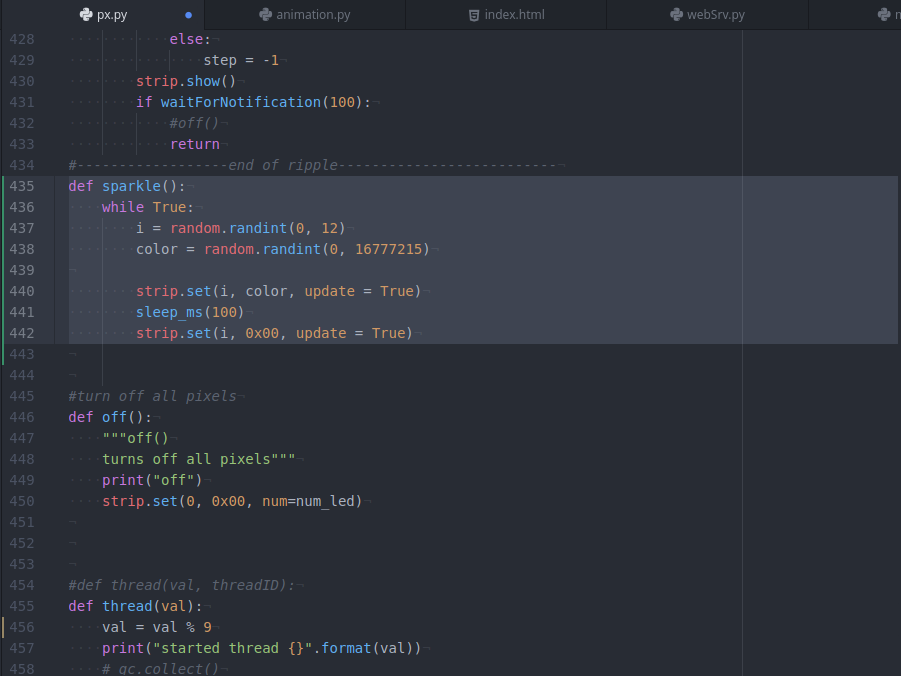
**www/index.html**, damit wir die Animation auch auf der Website auswählen können

**main.py** um die Eingaben der Website entgegenzunehmen und die Animation in einem Task zu starten.

Fangen wir mit der File px.py an:

Wenn Du die Variablennamen genauso übernommen hast, wie sie beschrieben wurden, kannst Du einfach die Funktion sparkle() komplett kopieren und in px.py einfügen.

Die Datei ist schon ziemlich lang, daher würde ich vorschlagen, Du fügst sie am relativ am Ende, noch vor der Funktion off() ein:

****

Eine kleine Änderung an der Funktion müssen wir allerdings vornehmen.

Die Pixelanimationen laufen in einem anderen Task wie der Webserver. Dazu gibt es noch einen separaten Task für die main.

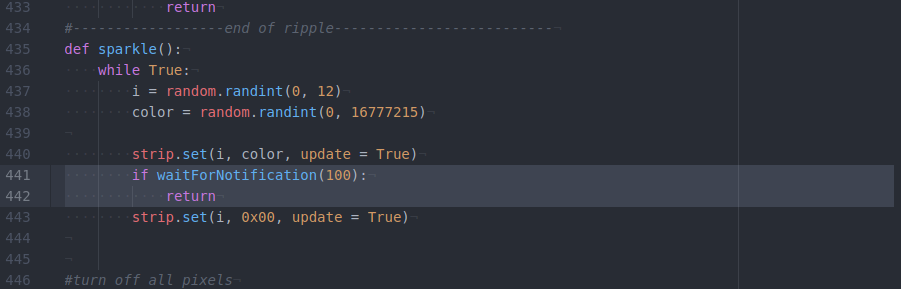
Damit die Main die Animation abbrechen kann, müssen die Tasks miteinander kommunizieren.

Eine Funkton, die das erledigt, ist bereits vorhanden. Diese befindet sich ab Zeile 23 und nennt sich waitForNotifications.

Hier wird einfach nur der Task für die angegebene Anzahl an Millisekunden ausgesetzt und während dieser Zeit auch geprüft, ob Benachrichtigungen vorhanden sind. \_thread.wait(timeout) erledigt das für uns.

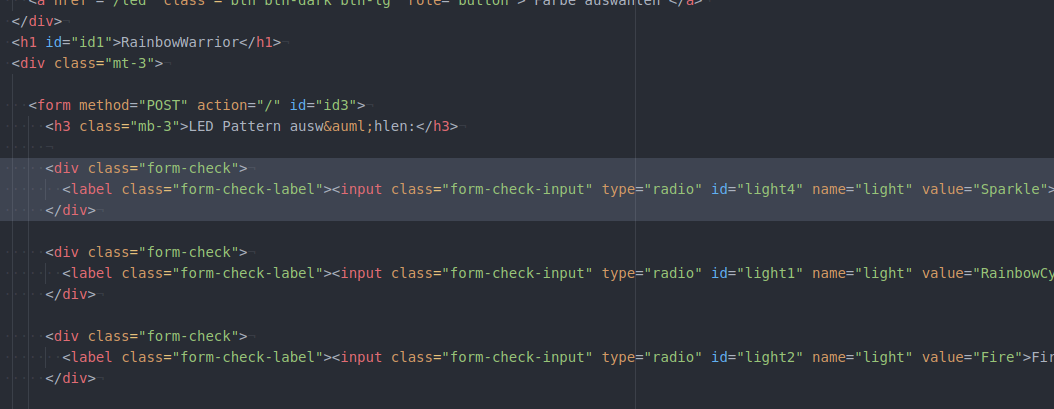
Wenn eine Benachrichtigung gekommen ist (\_thread.EXIT wenn die Benachrichtigung vom Mainthread kam, 666 wenn sie vom Server kam), wird der wert „True“ zurück geliefert, andernfalls der Wert „False“. Also muss die Funktion in eine Bedingung gepackt werden, die die Endlosschleife verlässt, wenn die Abbruchbedingung gegeben ist. Andernfalls wird die Schleife weiter ausgeführt.

Da die Ausführung der Schleife durch die Funktion sleep\_ms(200) ebenso ausgesetzt wird, kann man diese einfach durch waitForNotifications(200) ersetzen:



Als nächstes öffnen wir die Datei „index.html“, die sich im Ordner „www“ befindet.

Hier müssen wir dafür sorgen, dass die Funktion, ebenso wie die anderen, mit einem Radio-Button auf der Website angezeigt wird:



Da dieser Part sehr lang ist, kannst Du ihn Dir auch kopieren:

**<div class="form-check">**

**<label class="form-check-label"><input class="form-check-input" type="radio" id="light4" name="light" value="Sparkle">Sparkle</label>**

**</div>**

Jetzt haben wir also unsere Funktion eingefügt, eine Möglichkeit geschaffen, diese aufzurufen. Bevor dies allerdings funktionieren kann müssen wir noch die aufgefangenen Daten in der main verarbeiten.

Wichtig in dem eben eingefügten Code ist vor allem der Wert(value), also „Sparkle. Dieser wird aus der http-Form mittels Thread Messages weiter gegeben (was in webSrv.py, Zeile 33 bis 45, geschieht) und sollte nun in der main.py verarbeitet werden.

Auf Zeile 209 wird mit **msg = \_thread.getmsg()** geprüft, ob der Webserver neue Daten per Message geschickt hat.

Wenn es sich um eine neue LED-Animation handelt, sieht die Nachricht so aus:

**( {message-type}, {ID des Absenders}, „light:Sparkle“ )**

Also muss als erstes geprüft werden, ob es sich bei der Nachricht um einen String, also einen Zeichenkette handelt. Dies ist der Fall, wenn message-type = 2 ist. Außerdem muss die ID des Absenders geprüft werden. Die ID des server-threads wird in der Variable „srv\_thread“ gespeichert. Die Bedingung

**If (msg[0] == 2 and msg[1] == srv\_thread)**

erledigt dies für uns. Ist diese Bedingung erfüllt, wird die Nachricht, die in msg[3] gespeichert ist, am Doppelpúnkts gesplittet:

**values = msg[2].split(":")**

Nun muss geprüft werden, ob auch wirklich das Licht angeschalten werden muss:

**values = msg[2].split(":")**

Erst dann kann geschaut werden, welche Animation gestartet werden muss indem der Name geprüft wird.

Jeder Animation ist ein Zahlenwert zugewiesen. Dies ist leider etwas umständlich, aber nur so konnte ich einfach gewährleisten, dass durch Berührung des Touch-Sensors ebenso die Animationen gesteuert werden können.

So muss bei Berührung nur die Zahl, die in „light\_val“ gespeichert wurde, um eins erhöht werden. Wenn sich die light\_val dann von der vorher gespeicherten „last\_light\_val“ unterscheidet, kann in „handleLightThread(val)“ die eventuell noch laufende Animation abgebrochen werden und eine neue gestartet werden.

Um also dafür zu sorgen, dass die Animation „Sparkle“ sowohl durch den Webserver als auch durch den Touchsensor ausgewählt werden kann, muss auch ihr ein Zahlenwert zugewiesen werden.

Die Message vom Server wird abgefragt auf Zeile 209 in main.py.

Hier fügst du dann in die Bedingungen die Zeilen

**if "Sparkle" in values[1]:**

**light\_val = 9**

ein.



Auf Zeile 261 in main.py wird geprüft, ob sich der light\_val verändert wird und anschließend die Funktion „handleLightThread()“ auf Zeile 81 aufgerufen.

Nachdem ggf. eine laufende Animation abgebrochen wurde muss auch hier der aktuelle Thread eine Pause einlegen. Die ist nur deshalb solange (3000 ms), damit die Website genug Zeit hat, neu zu laden. Wenn man hier nicht lange genug wartet, kann dies zu Systemabstürzen führen.

**lightAnim\_thread = \_thread.start\_new\_thread("lightThread", px.thread, (val,))**

Mit dieser Funktion wird ein neuer Task gestartet, in dem die Funktion thread(val) im Modul px läuft.

Also speichern wir die Datei main.py und öffnen erneut die Datei px.py.

Ganz am Ende dieser Datei befindet sich die Funktion thread(val), die dann die entsprechende Animation startet.

Hier fügen wir nun die Abfrage für unsere Funktion ein:

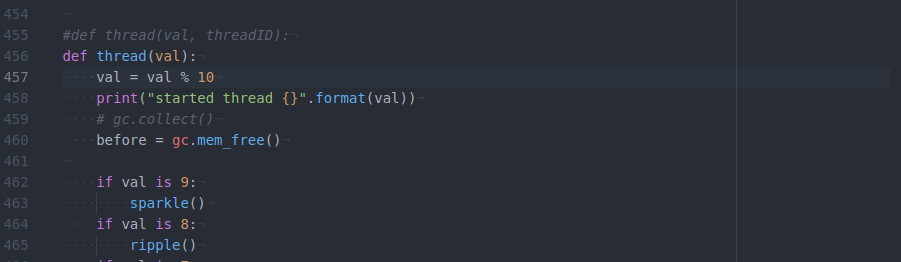
**if val is 9:**

**sparkle()**

Da aber der Touch-sensor immer nur den light\_val um eins erhöht, müssen wir gewährleisten, dass der eingehende Wert „val“ nie größer ist als die Anzahl der vorhandenen Threads.

Also müssen wir gleich nach Beginn der Funktion bei **val = val % 9** die 9 durch eine 10 ersetzen.

So sollte das ganze jetzt aussehen:



Zum Schluss müssen wir wieder im Terminal den Ordner mit unserem Code öffnen, mittels „rshell --buffer-size=30 -p /dev/ttyUSB0 -a -e nano“ aufs Board zugreifen und unsere Dateien mit dem bereits vorgestellten Befehl cp (cp main.py /flash) auf den Controller schieben.

Mit Repl geht’s weiter, anschließend entweder den „RST“ Pin auf dem controller oder mitteis „import machine; machine.reset()“ das Gerät neu starten.

Wenn alles gut gegangen ist, kannst Du die Animation nun auf der Website starten!