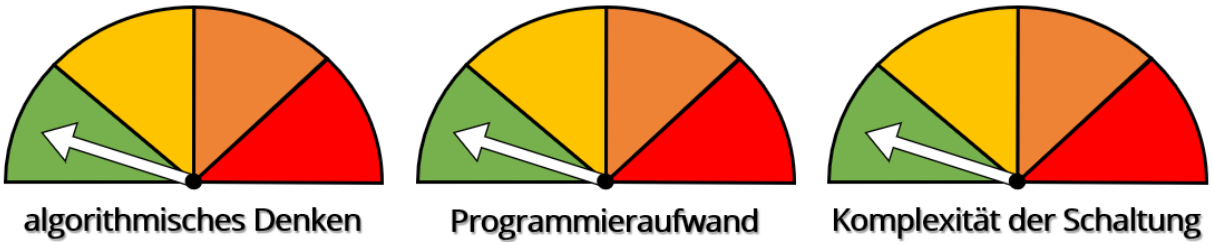
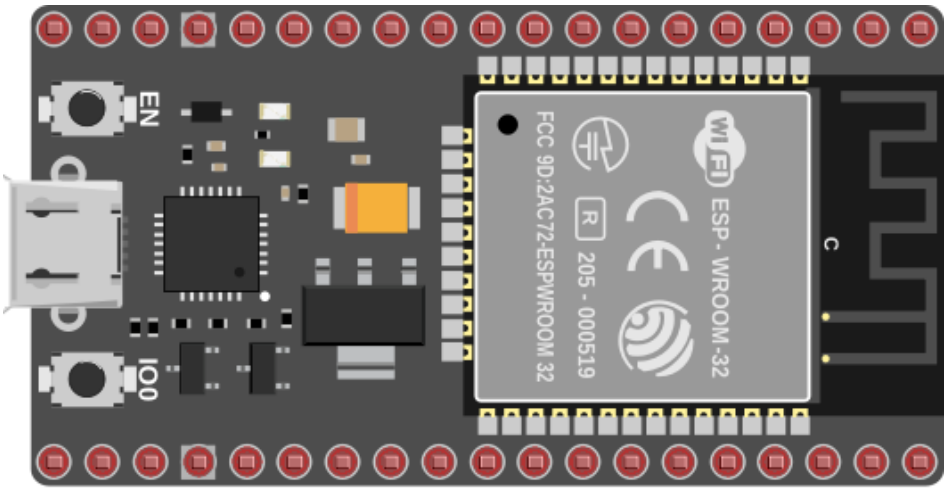
TECHNISCHE UNIVERSITÄT DRESDEN

MicroPython

Einstieg | Blinkende Lichter



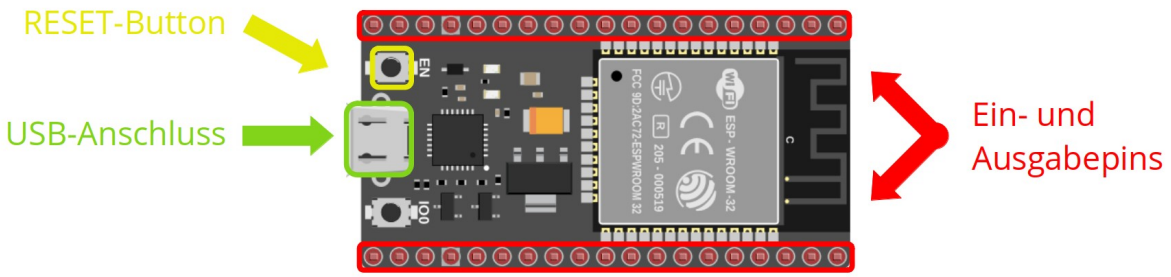


Professur für Didaktik der Informatik

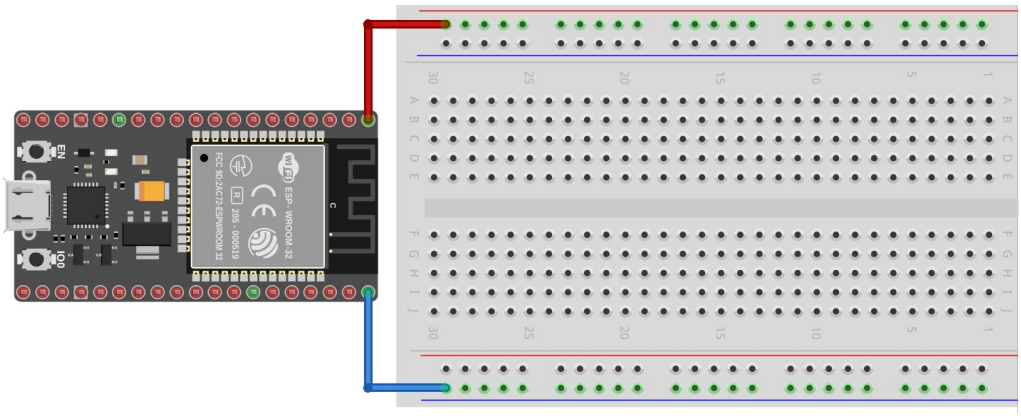
UM WAS WIRD ES IN DIESER STATION GEHEN?

Ob zu Hause, oder in der Schule, jeden Tag schaltest du Lichter ein und aus. Du siehst Blinklichter an Autos, Ampeln und der Baustelle. In dieser Station wollen wir genau diese Lichter nachbauen und programmieren. Doch zuerst lasst uns einen Blick auf das ESP32-Entwicklungsboard werfen.

DAS ESP32-ENTWICKLUNGSBOARD



USB-Anschluss	Der USB-Anschluss liefert den Strom für das Entwicklungsboard. Außerdem wird über ihn dein geschriebenes Programm auf dem Entwicklungsboard installiert.
Stromanschlüsse	Deine Bauteile auf dem Steckbrett brauchen Strom. Deswegen musst du sie mit den Stromanschlüssen verbinden.  **Minuspol (-)**  **Pluspol (+)**
Ein- und Ausgabepins	Diese Pins können Strom an- und ausschalten. Sie können aber auch lesen, ob am Ende der Verbindung Strom fließt.



Auf dem Steckbrett bringst du deine Bauteile an. Die äußeren Leisten (rot und blau) sind durchgehend miteinander verbunden. Verbindest du also ein Loch der äußeren blauen Leiste mit dem Minuspol, so sind auch die anderen Löcher der Leiste mit ihm verbunden. Für die mittleren Leisten gilt das Gleiche. Hier ist die Verbindung aber vertikal.

AUFGABE 1 – VERSORGE DAS STECKBRETT MIT STROM

Verbinde die äußeren Leisten mit den Stromanschlüssen, wie im oberen Bild. Der +Pol sollte immer mit der roten Leiste verbunden werden, der -Pol mit der blauen Leiste. Verwende dafür zwei beliebige Kabel.

NUN LASSEN WIR UNS EIN LICHT AUFGEHEN

Als nächsten Schritt, bringen wir eine LED zum Leuchten.

WAS IST EINE LED?

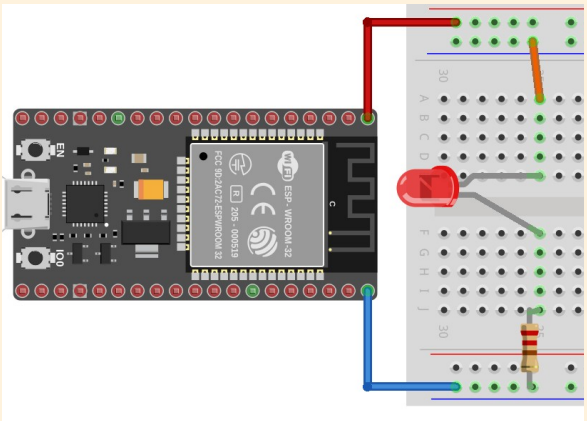
Eine **LED** ist eine Art kleine Lampe. Diese funktioniert jedoch nur, wenn du sie richtig herum anschließt. Bei genauerem Hinschauen siehst du, dass ein Beinchen länger ist, als das andere. Das lange muss mit dem Pluspol verbunden werden!
Würdest du nun die LED anschließen, würde sie kaputt gehen. Das liegt daran, dass sie alles an Strom aufnimmt, auch wenn sie dadurch überhitzt. Um das zu verhindern, brauchst du einen **Widerstand**. Baue diesen immer in deinen Stromkreis ein.

AUFGABE 2 – BRINGE DIE LED ZUM LEUCHTEN

Baue die LED und den Widerstand auf dem Steckbrett an. Verbinde sie wie auf dem Bild mit den Stromleisten.

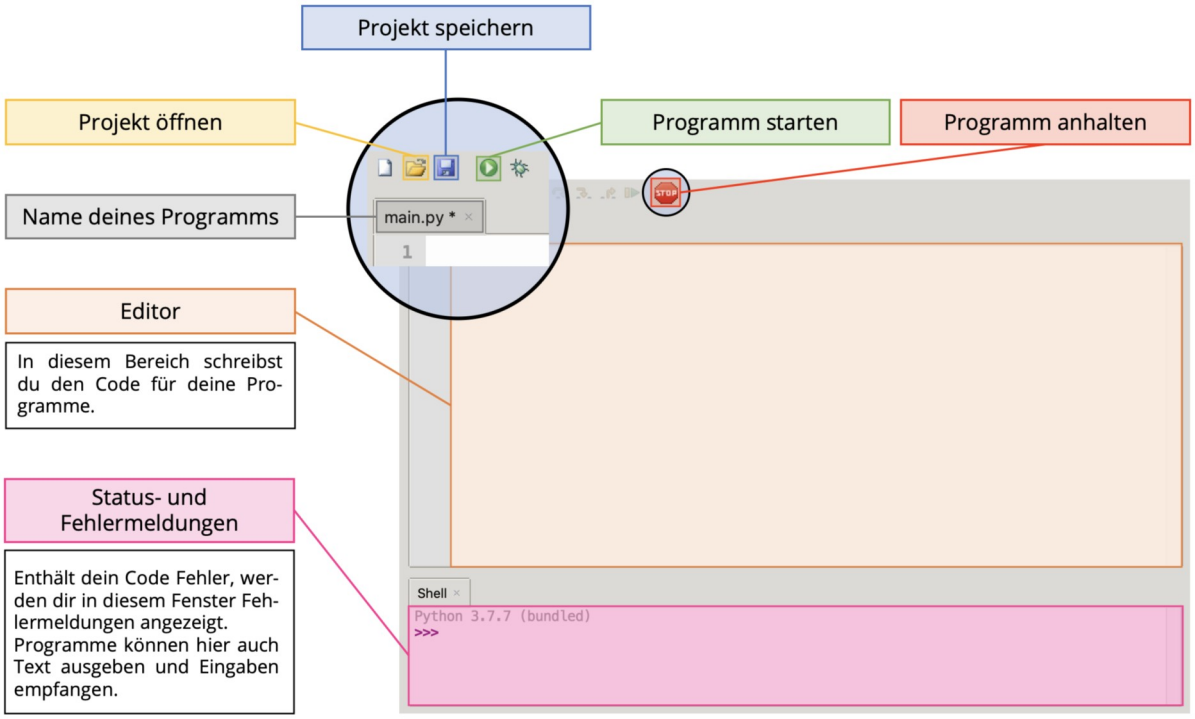
Alles verbunden? Dann verbinde jetzt das USB-Kabel mit dem Computer!

Die LED sollte jetzt leuchten!

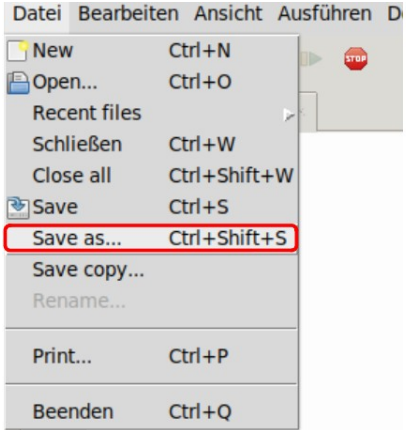


DIE LED DURCH DEN ESP32 STEUERN

Bisher haben wir die LED durch die Versorgung mit Strom zum Leuchten gebracht. Wir wollen aber nicht bei jedem Ein- und Ausschalten der LED die Kabel umstecken müssen. Unser Board soll den Strom automatisch ein- und ausschalten.
Deswegen brauchen wir die Programmierumgebung, um Code zu schreiben. Diese siehst du auf dem Bild unten.



Vor dem ersten Speichern musst du dem Computer sagen, dass du dein Programm auf dem Entwicklungsboard speichern möchtest. Wähle dafür unter *Datei* den Menüpunkt *Save as…* aus. Klicke in dem sich nun öffnenden Fenster auf *MicroPython device*.

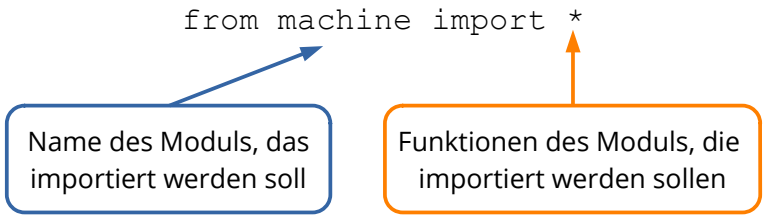


Um die Pins unseres Boards benutzen zu können, müssen wir ein Modul namens `machine` benutzen.

WAS IST EIN MODUL?

Ein **Modul** ist ein kleines Hilfsprogramm, das Funktionen enthält, die von anderen Programmen häufig benutzt werden. So befinden sich im Modul `machine` alle Funktionen, die du brauchst, um mit deinem Board kommunizieren zu können.

Module werden am Anfang von Programmen hinzugefügt. Man nennt diesen Vorgang **importieren**. Das Modul `machine` importierst du wie folgt:

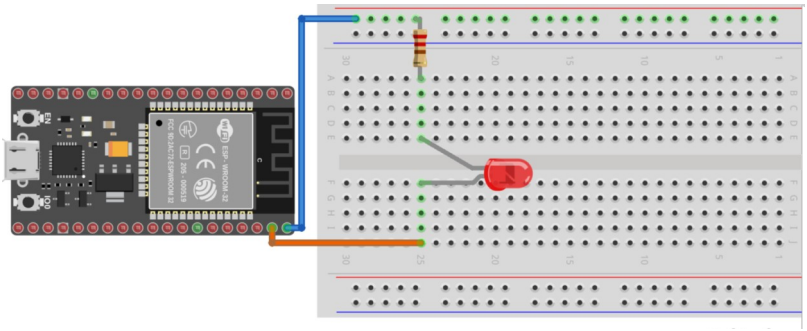


Der Stern `*` bedeutet hier, dass du alle Funktionen des Moduls `machine` in dein Programm importieren möchtest. Du kannst auch nur ganz bestimmte Funktionen aus einem Modul importieren. Dafür musst du den Stern durch den Namen der Funktion ersetzen.

AUFGABE 3 – LED GESTEUERT ZUM LEUCHTEN BRINGEN

Bringe die LED zum Leuchten, indem du sie mit dem ESP32-Board steuerst. Schreibe dafür einen passenden Code. Beachte dabei die folgenden Schritte.

1. **Die Schaltung abändern**
   Nimm das Kabel, das zuvor mit dem Pluspol verbunden war und stecke es in einen digitalen Pin – am besten in den **Pin 23**. Das lange Beinchen der LED sollte jetzt also mit einem Pin verbunden sein.

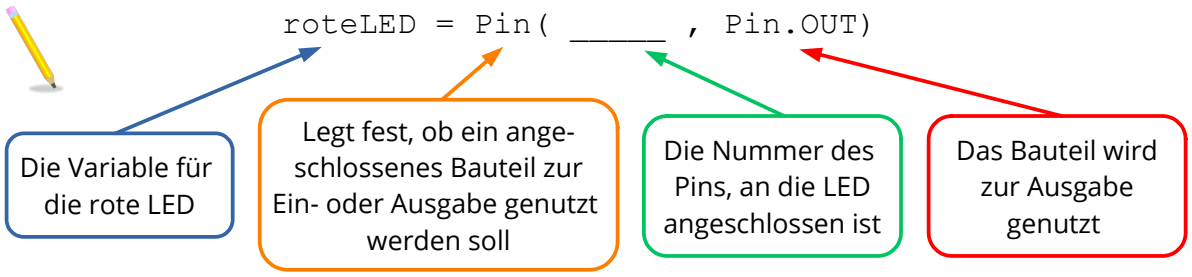


2. **Variable definieren und Pin verbinden**
   Als nächstes musst du dem Board sagen, an welchen Pin die LED angeschlossen ist. Dafür musst du eine Variable definieren.

WAS IST EINE VARIABLE?

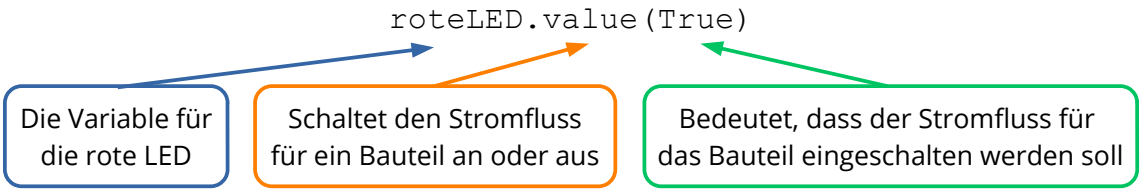
Eine **Variable** speichert eine Zahl, einen Buchstaben oder ein Wort unter einem bestimmten Namen, dem **Variablennamen**. Beim Programmieren ist das nützlich, weil man sich einfache Namen besser merken kann. Außerdem kann es vorkommen, dass sich Werte verändern – zum Beispiel die Pinnummer der LED, wenn wir sie an einen anderen Pin anschließen. Dann muss nur die Zahl in der Variablendefinition geändert werden und nicht jedes Mal die Zahl im Code ausgetauscht werden.

Wir geben der LED den Variablennamen `roteLED`.
Damit das Programm weiß, mit welchem Pin die LED verbunden ist, musst du der Variable `roteLED` den folgenden Wert zuweisen:



3. LED anschalten

Nun muss die LED nur noch eingeschaltet werden. Nutze dafür den Befehl



Der Wert `True` der Eigenschaft `value()` sagt dem Board, dass durch den Pin, mit dem die LED verbunden ist, nun Strom fließen soll – die LED leuchtet. Setzt du den Wert auf `False`, leuchtet die LED nicht.

4. Programm übertragen

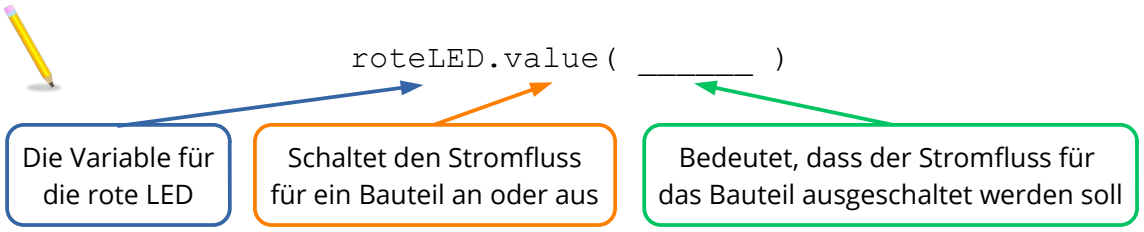
Falls du dein Programm jetzt zum ersten Mal speichern möchtest, beachte unbedingt den Hinweis hierzu auf Seite 5!
Hast du die Programmdatei als `main.py` auf dem Board gespeichert, kannst du nun auf den – Button klicken. Die LED sollte leuchten!

Dein Programm funktioniert nicht? Kein Problem, das ist ganz normal! Auf der letzten Seite findest du häufige Fehler und wie man sie beheben kann. Solltest du deinen Fehler nicht finden, frage ruhig eine Lehrperson.

AUFGABE 4 – LASSE DIE LED BLINKEN

Ändere deinen Code, damit die LED blinkt. Füge die nötigen Befehle an der richtigen Stelle im Code hinzu. Du findest sie unten. Installiere dein Programm auf dem Board und teste, ob die LED blinkt.

1. **Ausschalten**
   Um eine blinkende LED zu programmieren, müssen wir sie abwechselnd an und ausschalten. In unserem Code haben wir die LED bisher nur eingeschaltet. Den Befehl zum Ausschalten kennst du aber auch schon:



2. **Wiederholen**
   Damit unsere LED nicht nur einmal an- und ausgeschaltet wird, verwenden wir eine Schleife.

WAS IST EINE SCHLEIFE?

In Programmen kommt es häufig vor, dass Anweisungen wiederholt werden müssen. Dafür verwendet mal **Schleifen**.
Eine Schleife besteht aus den Anweisungen, die wiederholt ausgeführt werden sollen und einer **Abbruchbedingung**, die festlegt, wann nicht weiter wiederholt werden soll. Eine Schleife, bei der die Abbruchbedingung nie erfüllt wird, nennt man **Endlosschleife**.

In unserem Fall soll die LED so lange blinken, wie das Programm läuft. Wir müssen also eine **Endlosschleife** verwenden. Dafür benötigen wir eine Abbruchbedingung, die nicht erfüllt werden kann. Sie kann beispielsweise so aussehen:

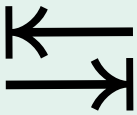
```
While True:
```

Diese Anweisung bedeutet auf Deutsch in etwa *„Solange das Programm läuft, mache …“*.

Nun fehlen noch die Anweisungen, die wiederholt werden sollen. Schreib sie in die Zeilen unter der Abbruchbedingung.
Aber Achtung: Die Grammatik von MicroPython verlangt, dass du die Anweisungen der Schleife unter der Abbruchbedingung mit der ⇆ - Taste einrückst!

GRAMMATIK VON PROGRAMMIERSPRACHEN

Wie in unsere Sprache, so hat auch eine Programmiersprache eine eigene Grammatik. Es ist wichtig die Regeln zu befolgen, sonst versteht dich der Computer vielleicht nicht. In Micro-Python ist es deswegen wichtig, dass du die Anweisungen einer Schleife mit der TAB-Taste einrückst. Sollte dir ein Fehler angezeigt werden, liegt das oft daran, dass ein Codeblock nicht korrekt eingerückt wurde.



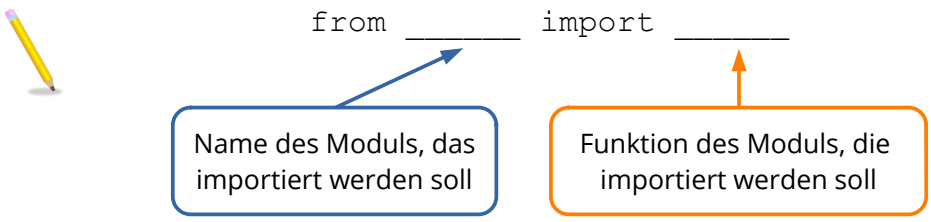
Was stellst du fest, wenn du die Befehle für das an- und ausschalten der LED in deinen Code geschrieben hast? Richtig! Die LED blinkt gar nicht ...
Das liegt an er Schnelligkeit unseres Entwicklungsboards. Es schaltet die LED so schnell nacheinander an und aus, dass das menschliche Auge es nicht sehen kann.

BEENDE DAS PROGRAMM AUCH IMMER

Solange das Programm auf dem ESP32 läuft kannst du kein neues Programm darauf speichern und abspielen lassen. Wenn du etwas am Code änderst drücke also erst auf die **Stop**-Taste im Thonny.

3. Verzögerung

Deswegen musst du dem Board sagen, dass es zwischen dem An- und Ausschalten eine Pause machen muss. Dafür müssen wir die Funktion `sleep` aus dem Modul `time` importieren:



Überlege, an welche Stelle im Code der Befehl eingefügt werden muss!

Anschließend kannst du das Programm mit dem Befehl

```
sleep(1)
```

für eine Sekunde pausieren. Überlege auch hier, wo der Befehl eingefügt werden muss!

AUFGABE 5 - WAHRNEHMUNG

Wie lange kann das Auge noch sehen? Verkleinere die Zahl im `sleep(1)` -Befehl solange, bis die LED nicht mehr blinkt. Erhöhe die Zahl dann langsam wieder, bis zu das Blinken wieder sehen kannst. So bekommst du den genauen Wert heraus. Kleiner als 1 ist zum Beispiel 0.8.

Ab welcher Zahl können deine Augen das Blinken nicht mehr sehen? Schreibe dein Ergebnis hier auf:

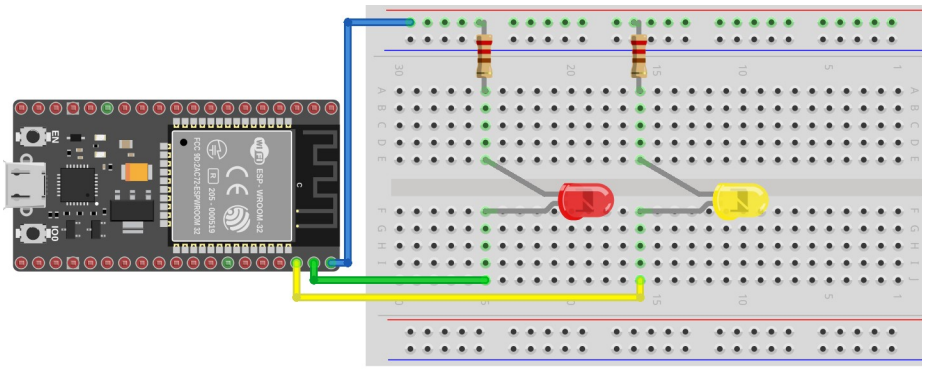
_____

AUFGABE 6 – WARNBLINKLICHT

Als nächstes wollen wir ein Warnlicht programmieren, wie man es von einer Baustelle kennt. Es besteht aus zwei Lichtern, die abwechselnd blinken. Folge dafür den nächsten Schritten.

1. Die Schaltung abändern

Verbinde die zweite LED mit dem Minuspol und dem **Pin 22** des Entwicklungsboards. Vergiss dabei nicht den Widerstand!



2. **Variable definieren und Pin verbinden**
   Auch für die gelbe LED musst du einen Variablennamen definieren. Du kannst sie `gelbeLED` nennen. Anschließend musst du ihr den Pin 22 des Boards zuweisen. Falls du nicht mehr weißt, wie es geht, schaue einfach nochmal in Aufgabe 3 oder deinem bereits geschriebenen Code nach.

3. **Absprechen der LEDs**
   Die rote LED geht bisher automatisch an und aus. Jetzt aber sollen sich die rote und die gelbe LED „absprechen“ und im Wechsel nacheinander leuchten. Damit sie wissen, wer an der Reihe ist, gibt es eine Art Schiedsrichter. Er wird durch eine Variable definiert. Zum Beispiel:

```
anderreihe = roteLED
```

   Damit legen wir fest, dass die rote LED zuerst an der Reihe ist.
   Wie muss der Befehl später aussehen, wenn die gelbe LED an der Reihe ist?

```
anderreihe = _____________
```

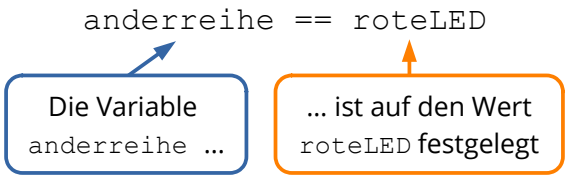
4. **Wenn und sonst**
   Wenn die rote LED an der Reihe ist, soll die rote LED leuchten. ***Sonst*** soll die gelbe LED leuchten.
   Beim Programmieren gibt es dafür diesen Ausdruck:

```
if rote LED ist an der Reihe:

    rote LED leuchtet

else:

    gelbe LED leuchtet
```

   Achte auch hier wieder unbedingt auf das Einrücken der Anweisungen nach den `if`- und `else`-Blöcken!

5. Befehle

Den Text im Beispiel kann das Entwicklungsboard natürlich noch nicht verstehen. Wir müssen ihn noch abändern.
Die rote LED ist an der Reihe, wenn die Variable `anderreihe` gleich `roteLED` ist:



Die rote LED haben wir schon zum Leuchten gebracht. Auch hier soll die LED eine Sekunde lang leuchten und sich dann wieder ausschalten. Die gleichen Befehle kannst du auch benutzen, um die gelbe LED eine Sekunde lang leuchten zu lassen.

Bringe die Codeteile in die richtige Reihenfolge!

```
while True:
    if anderreihe == _____:
        ________
        ________
        ________
    else:
        ________
        sleep(1)
        ________
        anderreihe = roteLED
```

A `roteLED.value(False)`

B `gelbeLED.value(True)`

C `sleep(1)`

D `roteLED.value(True)`

E `roteLED`

F `gelbeLED.value(False)`

Welche LEDs leuchten, nachdem du den Code eingefügt hast?

_____

6. **Schiedsrichterentscheidung**
   Dass nur die rote LED leuchtet liegt daran, dass der Schiedsrichter nie angesagt hat, dass die gelbe LED an der Reihe ist zu leuchten. Suche deswegen die Stelle, an der der Schiedsrichter diese Ansage machen muss. Und füge dort folgenden Befehl ein:

```
anderreihe = gelbeLED
```

Dein Programm funktioniert nicht?
Hier sind ein paar häufige Fehler und wie man sie beheben kann:

- **Die LED leuchtet nicht?**
  Überprüfe mithilfe der Bilder, ob die LED richtig angeschlossen ist. Das lange Beinchen muss immer mit dem Pluspol verbunden sein!

  Überprüfe, ob du die LED im Code auch tatsächlich angeschaltet hast und sich dahinter ein `sleep()` - Befehl befindet.

  Die Zeit im `sleep()` - Befehl entsprechen Sekunden. Diese sollte also größer als `0.01` sein, damit das menschliche Auge das Blinken wahrnehmen kann.

- **Du bekommst eine Fehlermeldung in der Programmierumgebung?**
  Überprüfe, ob du in Schleifen und Verzweigungen die Einrückungen eingehalten hast!

- **Nur eine der LEDs leuchtet?**
  Überprüfe, ob der Schiedsrichter auch ansagt, wann die neue LED blinken soll. Das heißt, am Ende jedes Blinken, muss die Variable `anderreihe` auf die neue LED gestellt werden. Bspw:

```
anderreihe = gelbeLED
```

Grafik Baustellenwarnlicht: *https://www.piqsels.com/* , *CC 0 1.0*
Screenshots: *fritzing electronics made by easy, Thonny 3.2.7 (Linux)*
Alle weiteren Grafiken: *Clemens Witt,* EduInf@TUD