1 Arbeiten mit der Onboard LED

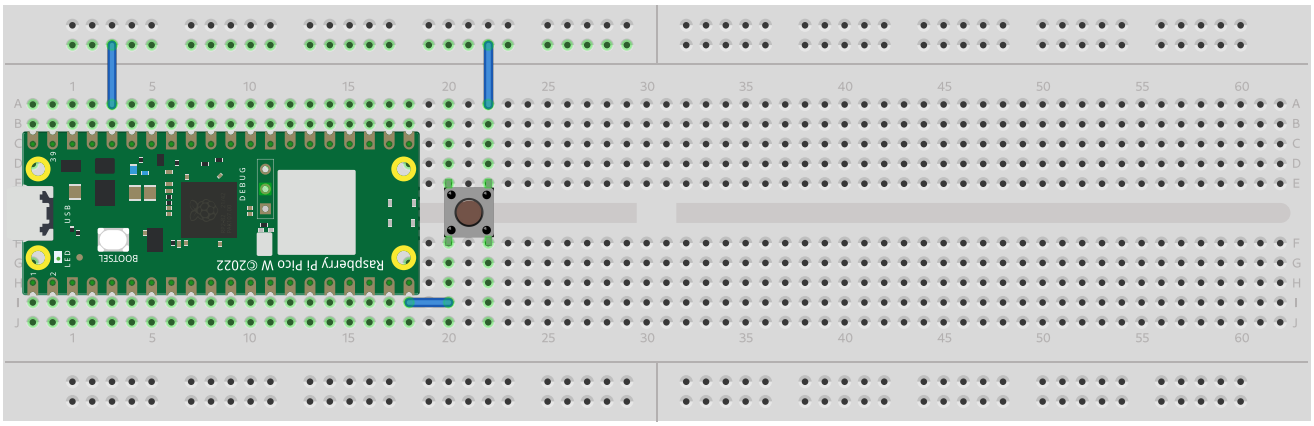
a) Öffne `01_onboard_led_task.py` und erledige das TODO.

b) Verwende Anstelle von `led.value()`, `led.toggle()`.

c) Erstelle eine Variable, mit der die Länge von `sleep()` angepasst werden kann.

2 Input

In der folgenden Abbildung ist eine Steckplatine zu sehen, in dem ein Taster an dem Pico angeschlossen ist. Baue dieses erst mal nach und mache danach mit den Codeaufgabe weiter.



a) Bearbeite die TODOs von `02_button_task.py` bis `04_button_task.py`.

3 Output

In der folgenden Abbildung ist eine Erweiterung der vorherigen Steckplatine um eine LED zu sehen. Baue dieses erst mal nach und mache danach mit den Codeaufgabe weiter.

a) Verändere bei den Dateien aus den vorherigen Aufgaben, den Pin so, dass die externe LED an- und ausgeschaltet wird.

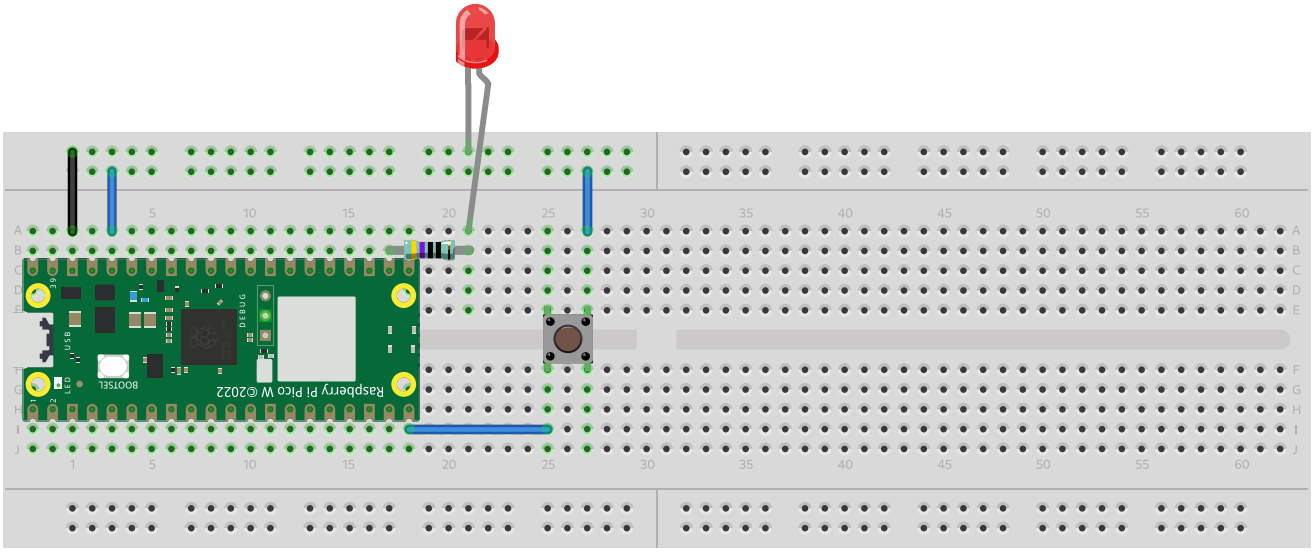
b) Öffne `05_led_task.py` und erledige die TODOs.

c) Werde kreativ und überlege dir noch ein weiteres Verhalten, dass die bisher eingeführten Komponenten verwendet!

Erweitern der Steckplatine

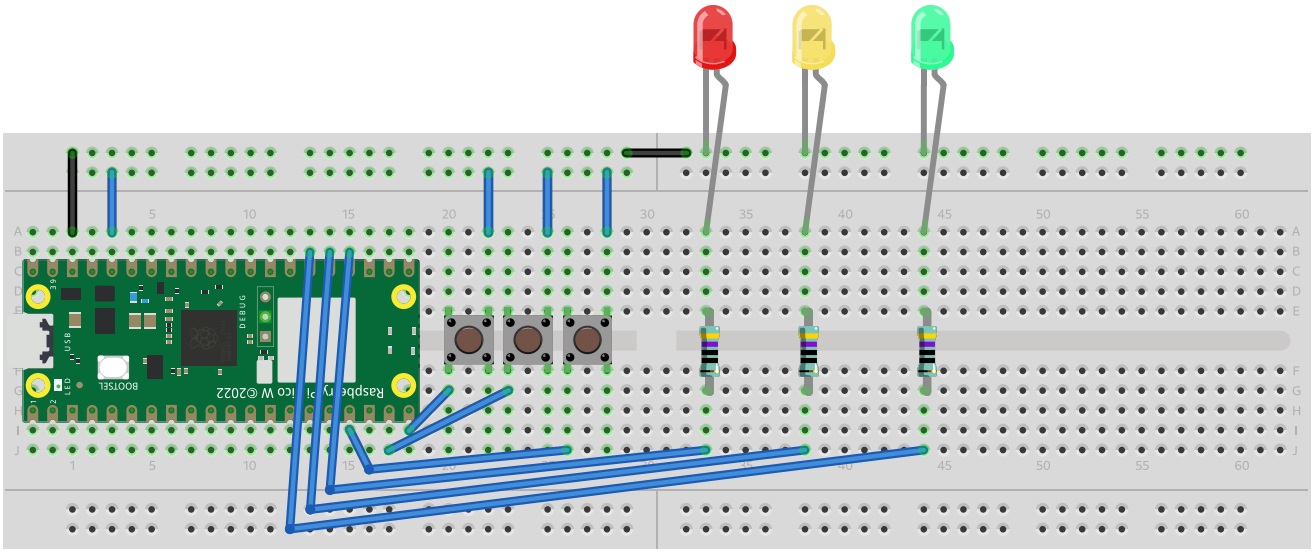
Im Folgenden kommen zwei größere Aufgaben, für die jeweils drei Taster und drei unterschiedlich farbige LEDs benötigt werden. Überlege dir, wie man die Steckplatine erweitern muss, damit diese drei Taster und drei LEDs enthält! Zu Beginn der nächsten Aufgabe ist eine mögliche Lösung zu sehen, falls du nicht weiter weißt.

Beispiele - CC BY Lehr-Lern-Labor Informatik



4 Lichtshow

In dieser Aufgabe wirst du eine Liste verwenden, um eine Reihenfolge von LEDs festzulegen, die der Pico auslesen und die entsprechenden LEDs aufleuchten lassen kann. Außerdem wirst du mehrere Möglichkeiten kennenlernen, neue Einträge in die Liste hinzufügen zu können. Für die Aufgabe sollte deine Steckplatine in etwa so aussehen:



Beispiele - CC BY Lehr-Lern-Labor Informatik

a) Öffne `06_lightshow_task.py` und erledige die TODOs, sodass die Lichtshow funktioniert.

- Für `play_light(light)` wird das übergebene `light` anschaltet, kurz gewartet und dann wieder ausschaltet. Fülle anschließend die Liste mit ein paar Einträgen und teste über die Konsole deinen Code!
- Bei `add_light(current_list, light)` wird ein gegebenes `light` an die gegebene Liste hinten anhängt. Hinweis: Listen haben eine Funktion `append(Eintrag)`, welche den übergebenen Eintrag hinten anhängen.
- Bei `add_random(current_list)` wird ein zufällig ausgewähltes Licht an die übergebene Liste anhängt.
- Für `add_per_button(current_list)` soll nach dem Aufruf auf eine Eingabe gewartet werden. Abhängig vom gedrückten Taster soll dann eine LED an die übergebene Liste dran gehängt werden. Am besten gibt man noch eine Rückmeldung aus, indem man beispielsweise die LED so lange leuchten lässt, wie der Taster gedrückt wird.

Wenn du nicht weiterkommst kannst du in `lightshow_tipps` noch ein paar Ansätze zum Vervollständigen finden!

5 Simon (Senso)

In dieser Aufgabe wirst du, aufbauend auf dem Wissen der Lichtshow, das Spiel Simon (Im Deutschen auch als Senso bekannt) implementieren.

Spielprinzip
Jeder Taster repräsentiert eine LED. Zuerst leuchtet eine zufällige Sequenz von LEDs auf. Danach wird auf die Eingabe des Spielers gewartet. Dieser muss durch Drücken der Taster die Sequenz von LEDs wiederholen. Gelingt dies dem Spieler, erhöht sich der Punkte-stand und der Sequenz wird eine weitere zufällige LED hinzugefügt. Wird die Sequenz nicht korrekt wiederholt, endet das Spiel.

a) Öffne `07_simon_task.py` und erledige die TODOs, sodass man das Spiel spielen kann.

- Es muss eine Liste von Lichtern geben, die abgespielt und referenziert werden kann.
- Für `show_lights(time_per_light)` wird die Liste von Lichtern abspielt, wobei jedes Licht für die Länge `time_per_light` leuchten soll.
- Bei `fail()` wird der Spielzustand auf nicht spielend geändert und den Punkte-stand ausgegeben. Am besten wird hier auch auf irgendeine Art und Weise visualisiert, dass das Spiel vorbei ist.
- Für `repeat_lights()` wird für jede LED in der Liste auf eine Eingabe gewartet gewartet. Wird eine Eingabe gedrückt, so wird geprüft, ob diese mit der LED der Liste übereinstimmt. Ist dies der Fall, geht es für die nächste LED weiter. Andernfalls wird `fail()` aufgerufen.
- Mit `play()` wird das Spiel so lange gespielt, wie der Spielzustand auf spielend ist. Hier ist auch eine geeignete Stelle, um den Punktestand zu erhöhen und die Sequenz um eine LED zu erweitern
- Bei `restart()` werden der Spielzustand, der Punkte-stand und die Sequenz der LEDs zurückgesetzt.

Beispiele - © Lehr-Lern-Labor Informatik

6 Zusatz: Morsecode

In dieser Aufgabe kannst du am Beispiel von Morsecode mit python dictionaries arbeiten.

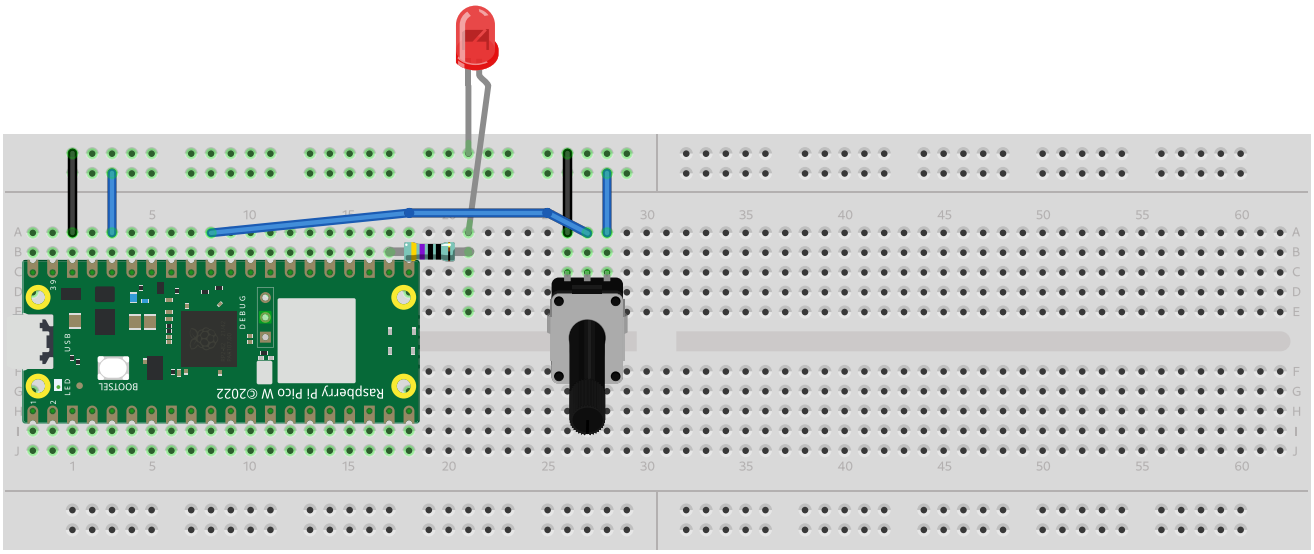
a) Öffne `08_morse_task.py` und betrachte den vorhandenen Code. Setze die richtige Nummer für den definierten Pin. Wie funktioniert deiner Meinung nach `morse_dict`?

b) Implementiere `translate_to_morse(sentence)`, welches einen gegebenen Satz in einen bestehend aus Morsecode umwandelt. Benutze dafür `morse_dict`!

c) Abschließend implementierst du `translate_to_morse(sentence)` welches einen eingebenden Satz als Morse über das Blinken einer LED ausgibt.

7 Zusatz: Analoger Input und Output

Bisher hast du mit digitalen Ein- und Ausgaben gearbeitet, welche nach dem Prinzip ganz oder gar nicht arbeiten. Mit dem Pico lassen sich aber analoge Werte Ein- und Ausgeben lassen, was das Arbeiten mit Zwischenwerten ermöglicht. Übernehme zu Beginn die wie folgt abgebildete Steckplatine:



a) Öffne nun `09_pwm_led_task.py` und betrachte den vorhandenen Code. Setze die richtige Nummer für den definierten Pin. Probiere über die Konsole die vordefinierten Methoden aus.

b) Implementiere die TODOs in den vorhandenen Schleifen so, dass die LED aus und angeht. Sobald es funktioniert, kannst du mit der Frequenz herumspielen. Was kannst du beobachten?

c) Öffne als nächstes `10_pwm_led_poti_task.py` und führe den Code aus. Drehe am Potenziometer, was kannst du beobachten?

d) Implementiere das TODO.

Beispiele - CC BY Lehr-Lern-Labor Informatik