Projekt zu Micropython: Taster

Sebastian Stigler

1 Taster entprellen

Tabelle 1: Zuordnung der Taster zu den GPIOs am Microcontroller.

Taster	GPIO
(U) Oben	2
(D) Unten	13
(L) Links	12
(R) Rechts	14

1.1 Einfaches Entprellen der Taster

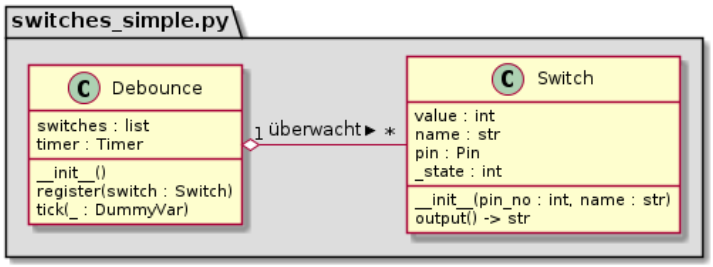

Abbildung 1: switches_simple.py

Schreiben Sie eine Wrapperklasse `Switch` um die eingebaute `Pin` Klasse mit folgenden Eigenschaften:

Attribute:

`value` speichert den entprellen Wert des Tasters.

`name` speichert die Kurzbeschreibung des Pins (U, D, L, R) als String.

`pin` ist ein `Pin` Objekt, das als Input mit internen Pull Ups definiert ist.

`state` speichert die letzten 12 tatsächlichen Werte von `Pin` in einer einzigen Variable (Tipp: benutzen Sie in die `<<`, `|` und `&` Operatoren).

Methode:

`output()` gibt bei gedrücktem Taster den Wert von `name` zurück. Bei geöffnetem Taster einen Punkt.

Schreiben Sie eine zweite Klasse `Debounce`, die das setzen der entprellten Werte nebst dem setzen der historischen Werte in **allen** `Switch`-Objekten übernimmt. Von der `Debounce` Klasse wird nur eine Instanz erstellt, die alle `Switch`-Objekte verwaltet.

Attribute:

`switches` ist eine Liste für die `Switch` Instanzen, die registriert wurden.

`timer` ist ein `Timer` Object, das den Takt für das Einlesen der tatsächlichen `Pin` Werte erzeugt.

Methoden:

`register(switch)` wird benutzt um ein `Switch` Objekt zu registrieren.

`tick(_)` dient als Callback Methode für den `Timer`. Hierin werden die historischen Werte in den registrierten `Switch`-Objekten gesetzt, ausgewertet und ggf. deren `value` Attribute verändert.

Testen

Testen Sie Ihren Code mit `joystick_full_loop.py`.

Die Datei `switches_simple.py` enthält die Klassen `Switch` und `Debounce`.

1.2 Erweitern von `Switch` um Callbacks für steigende und fallende Flanken

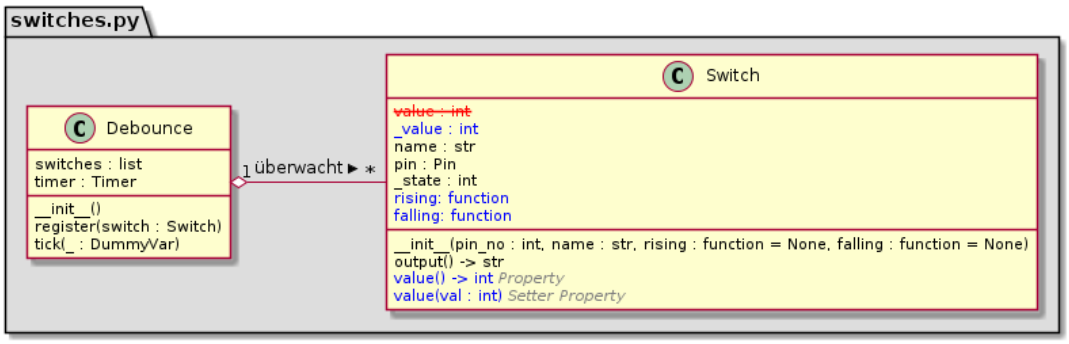

Abbildung 2: switches.py

Kopieren Sie die Datei `switches_simple.py` nach `switches.py`. Belassen Sie die Klasse `Debounce` so wie sie ist und ergänzen Sie in der Klasse `Switch` folgende Attribute:

Attribute:

`rising` speichert die Callbackfunktion für eine steigende Flanke (wenn der entprellte Wert von 0 nach 1 wechselt).

`falling` speichert die Callbackfunktion für die fallende Flanke.

Ändern Sie in `Switch` das Attribut `value` in eine Property mit Setter um die `rising` bzw. `falling` Callbacks auszuführen. Die Callbacks selbst haben **keine** Argumente!

Testen und Abgabe

Testen Sie Ihren Code mit `joystick_full_callback.py`.

Die Datei `switches.py` enthält die Klassen `Switch` und `Debounce`.

Ein Beispiel wie man aus einem Attribut eine Property mit Setter macht

```
class WithoutPropery:
    def __init__(self):
        self.x = 0

obj = WithoutProperty()
obj.x = 42
print(obj.x)  ## -> 42
```

```
class WithProperty:
    def __init__(self):
        self._x = 0  # rename var to _var

    @property
    def x(self):
        """Getter"""
        # do something
        return self._x

    @x.setter
    def x(self, value):
        """Setter"""
        # do something
        self._x = value

obj = WithProperty()
obj.x = 42
print(obj.x)  ## -> 42
```