Technische handleiding over Raspberry Pi Pico en microPython

Auteurs: [1]Angelika Tefelska, [1]Dariusz Tefelski Medewerkers:
[2]Chrissa Papasarantou, [2]Rene Alimisi

IoT4schools Consorcium:
[1]Technische Universiteit Warschau,
[2]EDUMOTIVA,

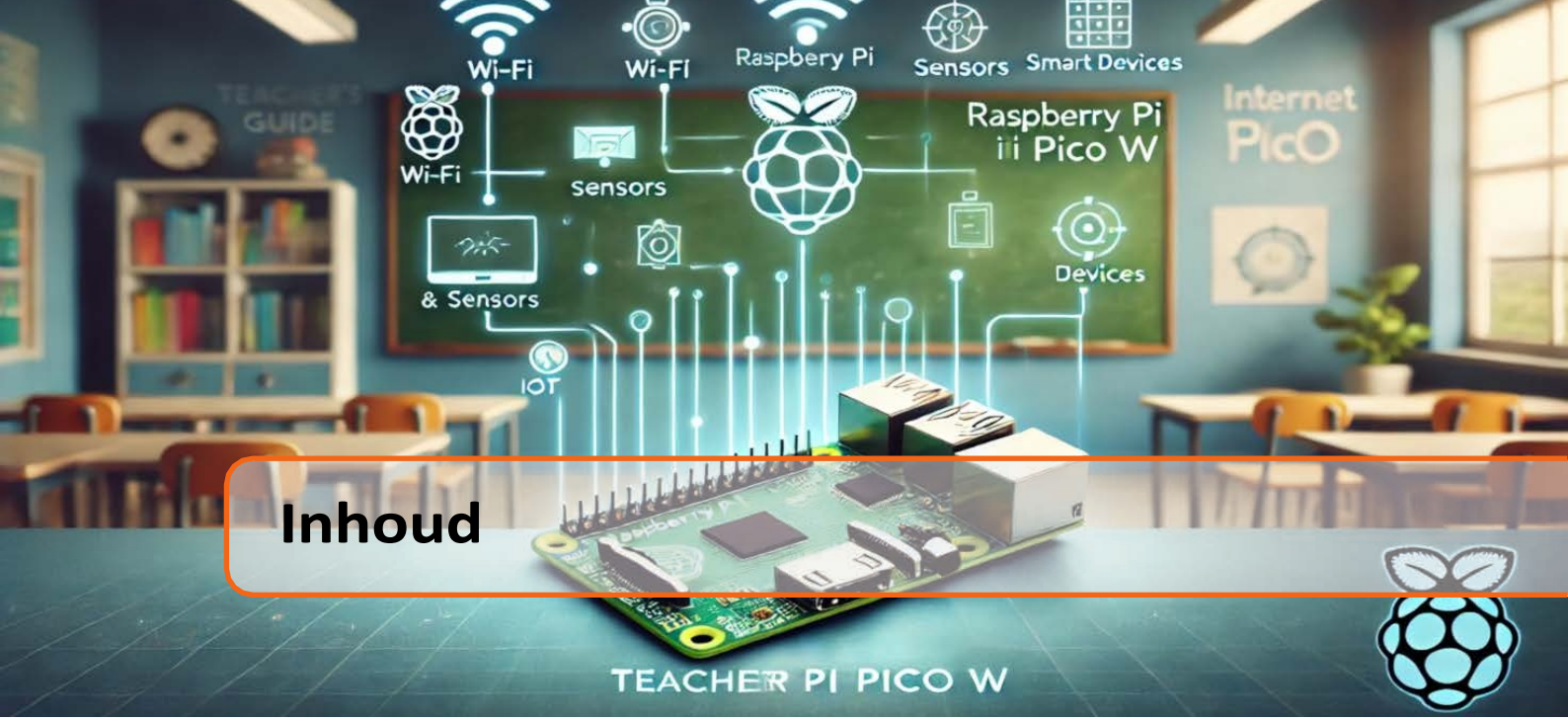
Projecttitel: "Bringing the Internet of Things in school education as a tool to address 21st century challenges", projectnummer: 2023-1-PL01-KA220-SCH-000154043, Erasmus+ KA220-SCH.

 LaTeX-sjabloon is overgenomen van:

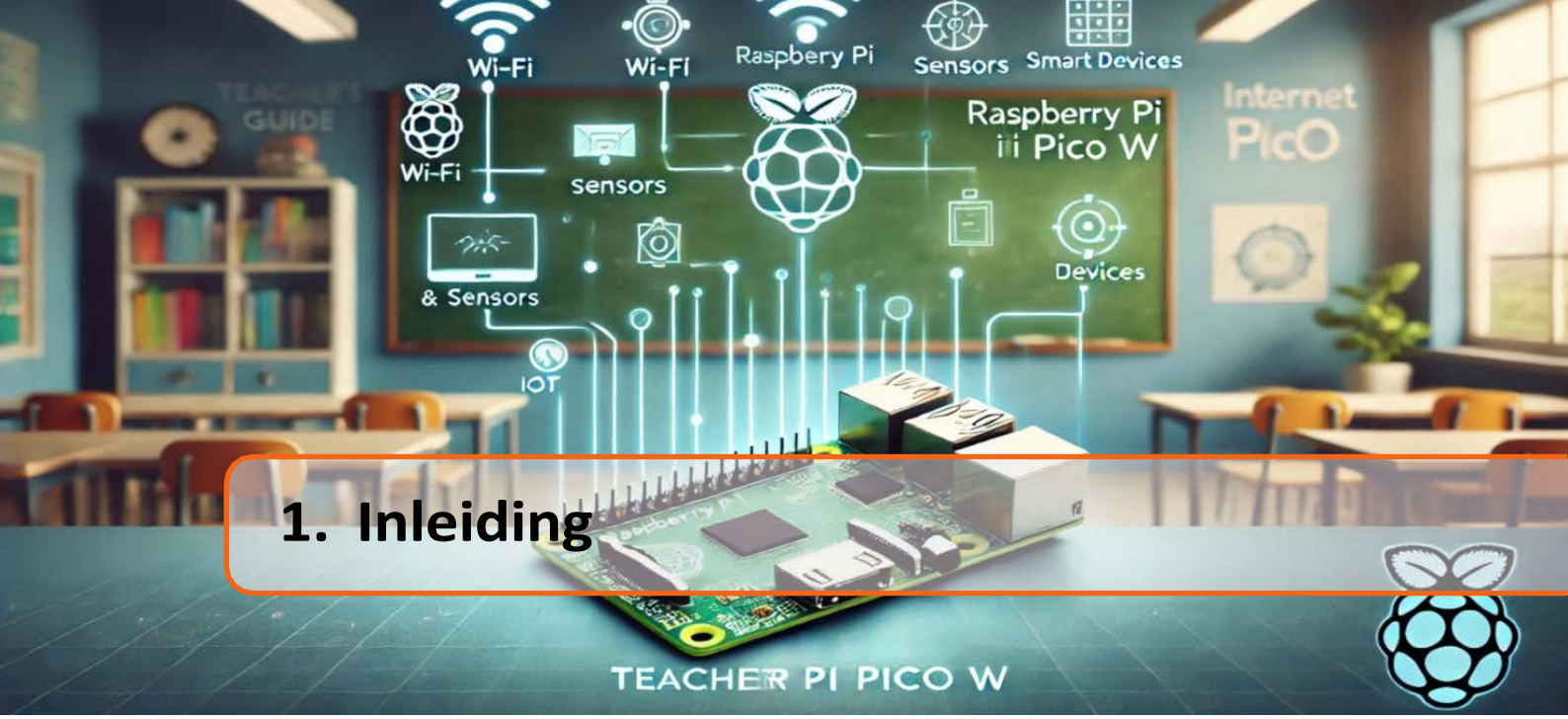
https://www.latextemplates.com/template/legrand-orange-book

Het hoofdstuk beelden gegenereerd met DALL-E, OpenAI.

Co-funded by the European Union



Inhoud

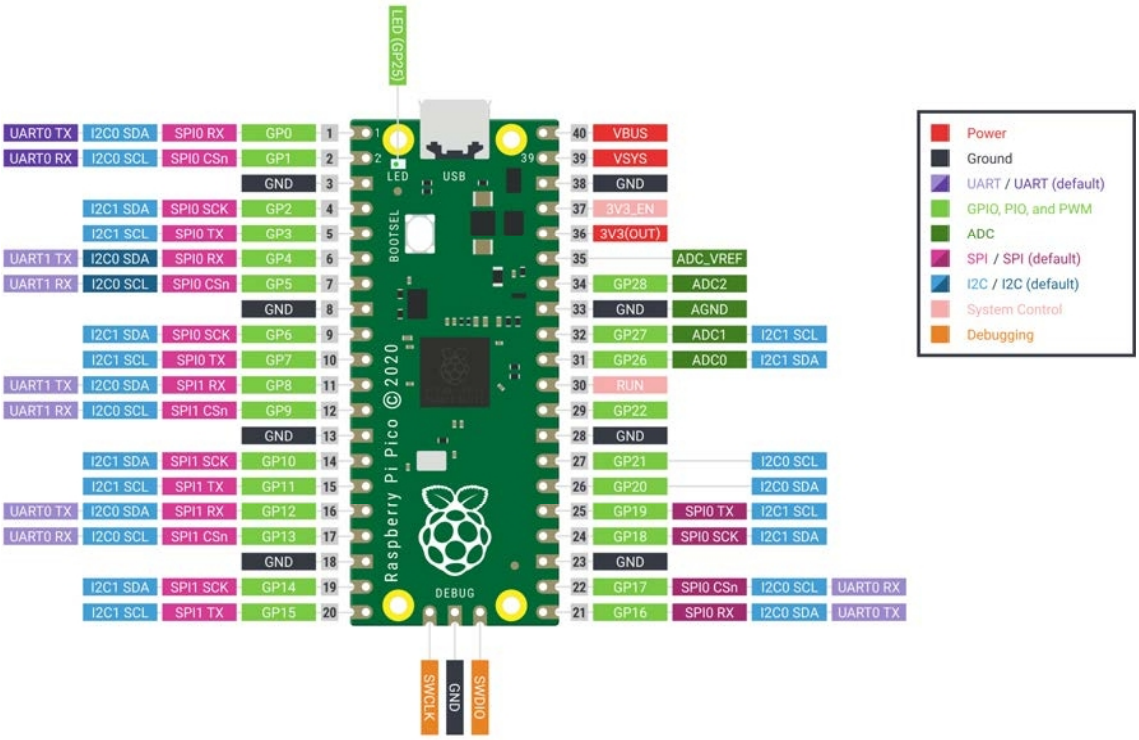


1. Inleiding

Deze handleiding is een introductie tot het Raspberry Pi Pico W board, dat gebruikt kan worden in Internet of Things (IoT) projecten. De ontwikkeling van digitalisering laat zien dat we steeds meer slimme oplossingen gebruiken die onze levenskwaliteit verbeteren en waarmee we hulpbronnen zoals elektriciteit of water kunnen besparen. Jezelf vertrouwd maken met het onderwerp IoT is uiterst belangrijk in de moderne wereld. Een van de borden waarmee je de wereld van IoT-technologie kunt betreden is de Raspberry Pi Pico W, die compact, gebruiksvriendelijk en goedkoop is. Vandaar dat deze gids zich richt op de presentatie van het Raspberry Pi Pico W-bord en de MicroPython-taal die wordt gebruikt om het te programmeren. De gids is bedoeld voor leerlingen, studenten, leraren, opvoeders en hobbyisten die nog niet eerder met het Raspberry Pi Pico W-bord hebben gewerkt. We moedigen je aan om te leren door oefening, d.w.z. door parallel voorbeeldprojecten uit te voeren met een gids, waardoor je snel vertrouwd raakt met het gebruik van het bord. Veel plezier en leerplezier!

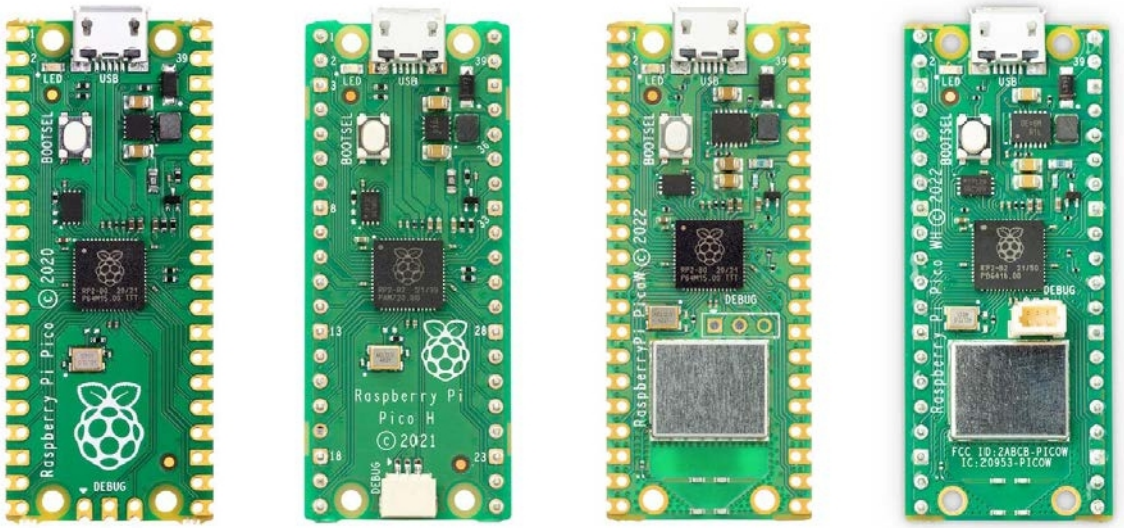
1.1 Raspberry Pi Pico bord

In 2021 werden de **Raspberry Pi** Pico-printplaten gelanceerd, een baanbrekend product van de Raspberry Pi Foundation. Eerdere versies van Raspberry Pi-boards waren System-on-Chip (SoC), wat miniatuurcomputers waren. De Raspberry Pi Pico serie is echter een heel ander type boards - ze zijn gebaseerd op microcontrollers en zijn een geweldig alternatief voor Arduino boards. Het hart van het Raspberry Pi Pico-bord is het eigen RP2040-systeem - een dual-core ARM Cortex M0+ microcontroller die werkt op 133 MHz, uitgerust met 264 KB SRAM en 2 MB Flash-geheugen. Het doel was om een efficiënte microcontroller te leveren met een aanzienlijke rekenkracht die zou voldoen aan de verwachtingen van hobbyisten over de hele wereld. Naast het uitstekende RP2040-systeem is het vermeldenswaard dat het bord was uitgerust met 26 GPIO-pinnen *(General-Purpose Input/Output*), waaronder 16 PWM-kanalen en communicatie-interfaces zoals I2C, SPI en UART. Als je niet bekend bent met de genoemde apparatuur op het bord, maakt dat niet uit, we zullen alle elementen later in de handleiding bespreken. De lay-out van de genoemde elementen op het Raspberry Pi Pico-bord is weergegeven in figuur 1.1.



Afbeelding 1.1: De pinout van Raspberry Pi Pico.Bron van afbeelding: https://www.raspberrypi.com.

De Raspberry Pi Pico serie bestaat uit 4 kleine printplaten (niet veel groter dan een pendrive) die je kunt zien in figuur 1.2.



Afbeelding 1.2: De 4 versies van Raspberry Pi Pico-borden. Bron van de afbeelding: https://www.raspberrypi.com.

Het eerste bord aan de linkerkant is de *Raspberry Pi Pico*, het volgende bord is de *Raspberry Pi Pico H*, die alleen verschilt doordat de pinnen al gesoldeerd zijn en een speciale connector hebben.

voor debugging-doeleinden is. Als je de standaardversie van de Raspberry Pi Pico bestelt, moet je de pinnen zelf solderen. Als je nog nooit gesoldeerd hebt, maak je dan geen zorgen. Op internet vind je veel voorbeeldvideo's over hoe je pinnen moet solderen, bijvoorbeeld op https://www.youtube.com/watch?v=zbhOyCA_4lg. Als je toch liever geen pinnen soldeert, is het beter om een bord te kiezen met de letter *H*. Houd er wel rekening mee dat je op dit bord geen pinnen voor debug-doeleinden kunt solderen en dat je een speciale connector moet gebruiken (kabel en connectoren verkrijgbaar bij het Raspberry Pi Debug probe-apparaat). Het derde bord van links is de *Raspberry Pi Pico W*, die bovendien een ingebouwde Wi-Fi- en Bluetooth-module heeft, wat erg belangrijk is in Internet of Things (IoT)-projecten. In deze handleiding zullen we ons vooral richten op de Raspberry Pi Pico W versie, die nodig is voor IoT-projecten. Het vierde bord is de *Raspberry Pi Pico WH*, die alleen verschilt van de Raspberry Pi Pico W in de pinnen die al gesoldeerd zijn (en de aanwezigheid van een debug probe connector). Als je niet van plan bent om pinnen te solderen, raden we daarom de Raspberry Pi Pico WH-versie aan voor IoT-projecten.

Alle Raspberry Pi Pico-borden zijn uitgerust met een microUSB-aansluiting, die gebruikt wordt voor het programmeren en voeden van het bord. Het bord kan geprogrammeerd worden in C/C++ of MicroPython. In deze handleiding richten we ons op MicroPython als de gemakkelijkste optie voor beginners. Voor degenen die Raspberry Pi Pico willen leren programmeren in C/C++ raden we de gids over dit onderwerp aan die hier beschikbaar is: https://datasheets.raspberrypi.com/pico/getting-started- with-pico.pdf.

Er zijn ook kaarten van derden op de markt, die meestal extra elementen en aansluitingen bevatten, waardoor bijvoorbeeld eenvoudig een batterij kan worden aangesloten, of een USB-C connector in plaats van een microUSB.

Nieuws 1.1.1

Onlangs is er een nieuwe versie van de Raspberry Pi Pico 2W uitgebracht, die vooral verschilt van zijn voorganger in: de microcontroller (de RP2040 gebaseerd op de ARM Cortex-M0+ Dual-Core 133 MHz is vervangen door de RP2350 gebaseerd op de ARM Cortex-M33 Dual-Core 150 MHz), de hoeveelheid SRAM en Flash geheugen is toegenomen (2x meer) en het aantal PWM kanalen is verhoogd van 16 naar 24. Als bonus verschenen er nieuwe kernen met RISC-V architectuur voor meer geavanceerde gebruikers als alternatieve keuze: ARM- of RISC-V-kernen, hoewel de ARM-kernen grotere mogelijkheden blijven bieden.

De nieuwe versie van de RP2350 microcontroller heeft echter één probleem met pull-down weerstanden (zie het hoofdstuk over digitale signalen). Ondanks het bovengenoemde probleem, dat kan worden verholpen, is de nieuwe versie een interessante keuze als je van plan bent uitgebreidere programma's te maken en meer geheugen nodig hebt. In deze handleiding wordt de Raspberry Pi Pico W gebruikt, wat voldoende is voor de meeste IoT-projecten.

1.2 Installatie

Voordat we beginnen met het programmeren van de Raspberry Pi Pico, moet je deze installeren:

- Thonny redacteur (https://projects.raspberrypi.org/en/projects/getting-begonnen-met-de-pico/2)
- Raspberry Pi Pico firmware (https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/3).

2. Elektronische basiscomponenten

Voordat we beginnen aan ons avontuur met het programmeren van de Raspberry Pi Pico W, moeten we eerst de elektronische basiscomponenten leren kennen die we gaan gebruiken:

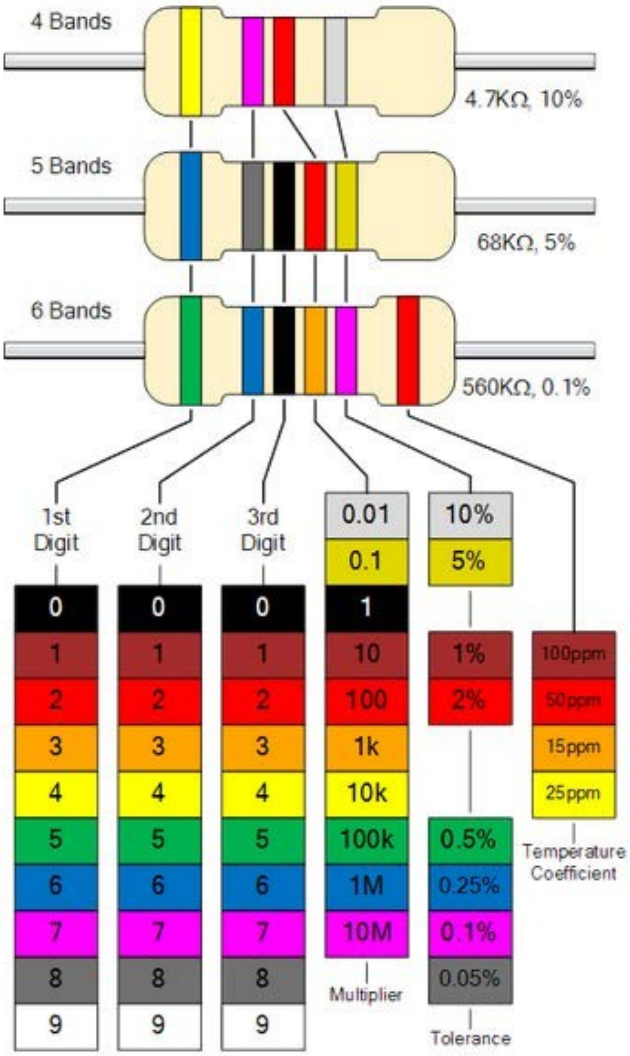
- **Weerstand** - is een element dat wordt gebruikt om de stroom die doorheen loopt te verminderen. Het dissipeert
  energie als warmte-energie. Een weerstand is een lineair element. Hoe groter de weerstand die we gebruiken, hoe minder stroom er door het circuit loopt, volgens de wet van Ohm:

$$I = \frac{V}{R} \tag{2.1}$$

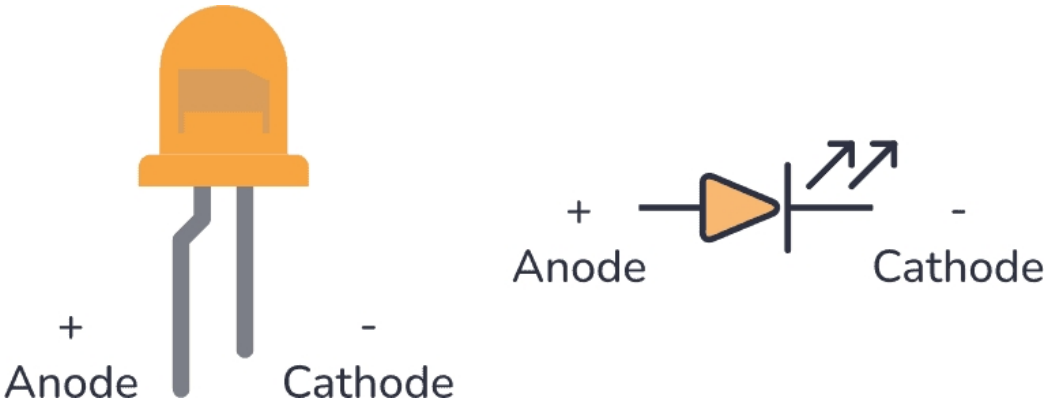
  waarbij: V - spanning op het element (in sommige landen gemarkeerd met de letter U), I - stroomsterkte, R - weerstand.
  De weerstandswaarde van een weerstand kan worden afgelezen van de kleurcode op de weerstand zoals in het voorbeeld in Fig. 2.1. Voor elke weerstand lezen we de weerstandswaarde af van links naar rechts. Bijvoorbeeld, in Fig. 2.1 heeft de bovenste weerstand de volgende bandkleuren: geel (waarde 4), roze (waarde 7), rood (x100) en zilver (tolerantie=10%). De waarde is dus 47x100Ω= 4700Ω=.*7kΩ* en tolerantie is 10%. De tolerantie geeft aan hoeveel de werkelijke weerstand kan afwijken van de nominale waarde (bijv. voor een weerstand van 10% 4,7kΩ zal de werkelijke weerstand tussen ,*23kΩ* (0,9 *47kΩ*) 5,17kΩ (1,- , *7kΩ*) liggen).
- **LED diode** - is een halfgeleiderelement dat stroom omzet in licht. Meer voor
  elektronen die van een hoger energieniveau naar een lager niveau gaan in een halfgeleider zenden een foton uit en afhankelijk van de energie van het foton krijgen we verschillende kleuren LED's. LED's zijn gepolariseerde elementen, wat betekent dat er maar in één richting stroom doorheen gaat. Fig. 2.2 toont een LED met een kathode- en anodedraad.

  De typische LED-diode heeft een spanning van ongeveer 1,7 V (V_{LED}) nodig en een stroom van 1 tot 15 *mA*. De spanning op de pinnen van de Raspberry Pi Pico is **3,3V** ($V_{(RP)}$), dus je moet een weerstand gebruiken om de stroom te beperken:

$$R = \frac{V_{RP} - V_{LED}}{I} = (107;\ 1600)\Omega \tag{2.2}$$

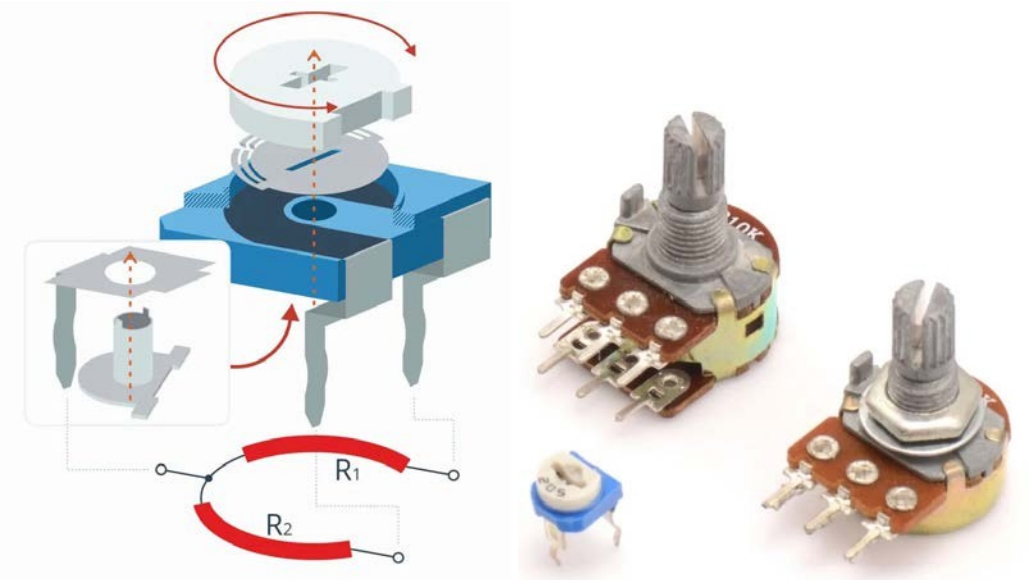


Afbeelding 2.1: De kleurcode van weerstanden. Bron: https://electronicspost.com/resistor-color-code/



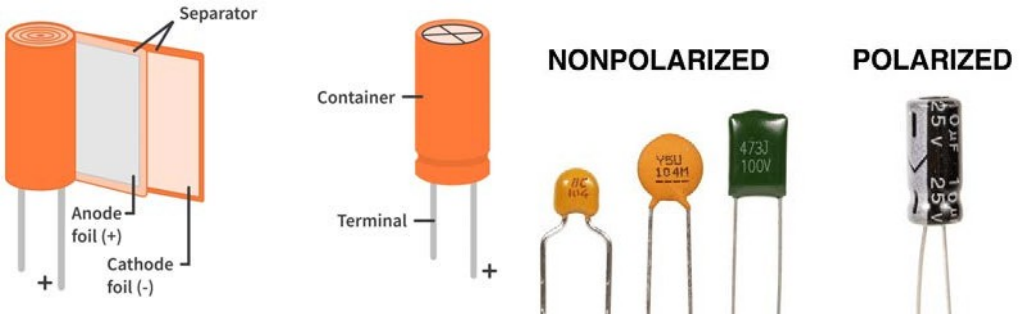
Afbeelding 2.2: Een LED met anode en kathode gemarkeerd (linkerafbeelding) en een diodesymbool (rechterafbeelding). Bron: https://www.build-electronic-circuits.com/what-is-an-l ed/ .

- **Potentiometer** - is een variabele weerstand waarmee je de weerstand tussen twee benen kunt aanpassen en zo de spanning kunt verdelen. Hij bestaat uit drie pootjes, waarbij twee pootjes zijn verbonden met het weerstandstraject (zie Fig. 2.3) en het derde pootje met de schuifregelaar. Door
  Door de positie van de schuifregelaar aan te passen, met een knop of een schroevendraaier afhankelijk van het type potentiometer, verdelen we het weerstandstraject in twee weerstanden die in serie zijn geschakeld. Op deze manier, als we de eerste poot verbinden met 3,3V en de derde poot met 0V, krijgen we een spanning tussen 0V en 3,3V op de middelste poot, afhankelijk van de stand van de knop.



Figuur 2.3: Werkingsprincipe van een potentiometer. Bron: https://forbot.pl/blog/leksykon/potencjometr .

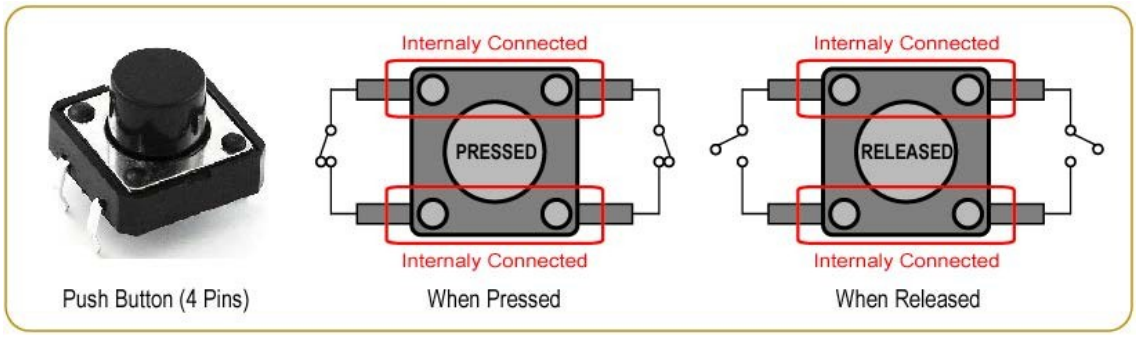
- **Condensator** - is een element dat een elektrische lading accumuleert. Het bestaat uit twee platen en een diëlektricum tussen de platen (zie linker Fig. 2.4). Condensatoren worden gebruikt,
  onder andere om een signaal te filteren (de condensator laat de gelijkspanning niet door, maar wel de rimpelingen, en kan de spanning dus afvlakken als hij met massa is verbonden) of om resonantiekringen te maken (een signaal met een specifieke frequentie opvangen).



Afbeelding 2.4: Afbeelding links: interne structuur van een condensator. Bron van de afbeelding: https://www.circuitbread.com. Rechter afbeelding: voorbeelden van gepolariseerde en niet-gepolariseerde condensatoren. Bron van de afbeelding: https://www.ariat-tech.pl

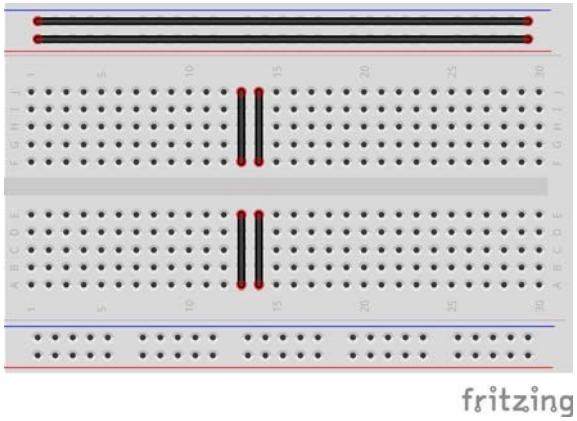
Condensatoren kunnen onderverdeeld worden in condensatoren die de juiste polarisatie vereisen (bijv. elektrolytische condensatoren) en condensatoren die dat niet vereisen (bijv. keramische of foliecondensatoren). Condensatoren die een juiste polarisatie vereisen, mogen slechts in één bepaalde richting aangesloten worden (zie de markeringen op de behuizing van de condensator). De elektrolytische condensatoren worden gekenmerkt door een hoge capaciteit, maar ze zijn niet efficiënt bij hoogfrequente signalen vanwege de dissipatiefactor. Keramische condensatoren drogen niet uit, maar ze zijn niet efficiënt bij het filteren van lage frequenties, omdat ze een kleinere capaciteit hebben.

- **Knop** - is een element dat het circuit sluit wanneer de knop wordt ingedrukt. Wanneer de knop niet wordt ingedrukt, is het circuit open. De constructie van de knop wordt getoond in Fig. 2.5.



Afbeelding 2.5: Werkingsprincipe van drukknoppen. Bron: https://gmostofabd.github.io /8051-Drukknop/.

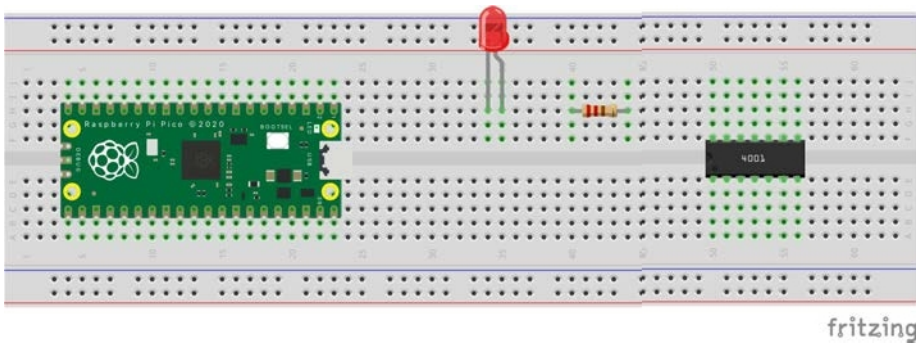
- **Breadboard** - is een soort printplaat die gebruikt wordt om elektronische componenten met elkaar te verbinden, d.w.z. om elektronische schakelingen te bouwen. Het bord bestaat uit vele gaten die verticaal met elkaar verbonden zijn, zoals te zien is in Fig. 2.6 (zwarte lijn).



Afbeelding 2.6: Aangesloten gaten zijn gemarkeerd met zwarte lijnen op het breadboard.

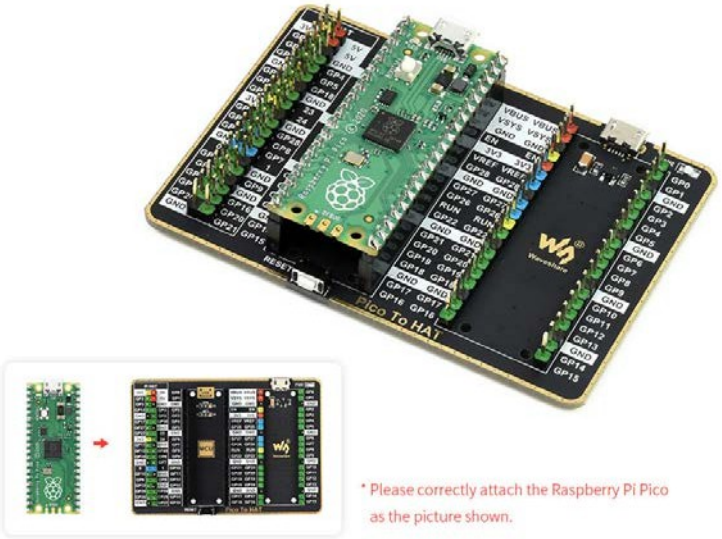
De gaten zijn dus in kolommen verbonden, maar niet in rijen. Het bord bestaat uit twee delen die gescheiden worden door een grote opening. Deze twee delen zijn niet met verbonden. Aan de randen van de printplaat is een sectie voor het aansluiten van voeding en aarde. Het is meestal gemarkeerd met twee lijnen: rood en blauw. In dit geval zijn de gaten horizontaal verbonden en niet verticaal. Dat wil zeggen, langs de rode lijn zijn alle gaten verbonden en hier sluiten we meestal de voeding aan, dat wil zeggen 3,3V in het geval van de Raspberry Pi Pico. Op dezelfde manier zijn alle gaatjes langs de blauwe lijn aangesloten en sluiten we meestal

Sluit daar massa aan. Fig. 2.7 toont de juiste plaatsing van voorbeeldcomponenten zodat hun pinnen niet worden kortgesloten.

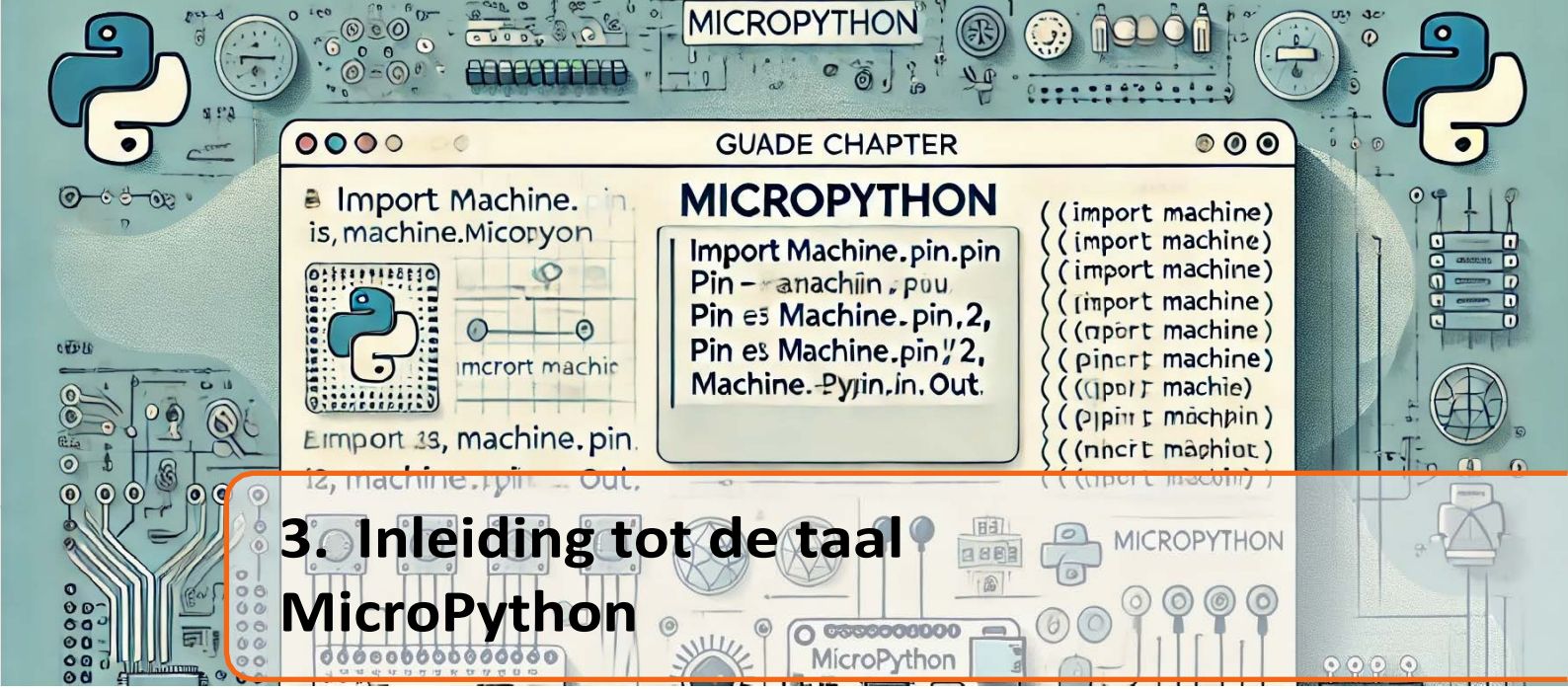


Afbeelding 2.7: Voorbeeld van het plaatsen van elementen op een breadboard zodat de pootjes niet worden kortgesloten.

- **Sensoren** - zijn elementen waarmee verschillende fysische grootheden kunnen worden gemeten, Bijvoorbeeld temperatuur, druk, vochtigheid, afstand, beweging, hartslag, enz. Sensoren geven gemeten waarden terug als: analoge signalen of digitale signalen die gewoonlijk worden verzonden via datacommunicatie-interfaces (bus).
- **Actuatoren** - zijn elementen die beweging uitvoeren, bijv. servomechanismen, DC motoren.
- **Shields** - zijn borden die de functionaliteit van het basisbord uitbreiden. Er zijn vele soorten shields. In het begin is een zeer nuttig shield de GPIO Expander For Raspberry Pi Pico (bijvoorbeeld van Waveshare *Pico naar hoed* - zie Fig. 2.8), waarmee u componenten op de Raspberry Pi Pico kunt aansluiten zonder dat u de Raspberry Pi Pico op het breadboard hoeft te plaatsen. Deze oplossing vermindert het risico op beschadiging van het Raspberry Pi Pico-bord door het verkeerd op het breadboard te plaatsen.



Figuur 2.8: Waveshare Pico to hat expander. Bron: https://www.waveshare.com/pi co-to-hat.htm



3. Inleiding tot de taal MicroPython

Dit hoofdstuk behandelt een aantal basiselementen van de MicroPython-taal, die nodig zijn om eenvoudige projecten te maken op basis van de Raspberry Pi Pico. Dit zijn ze:

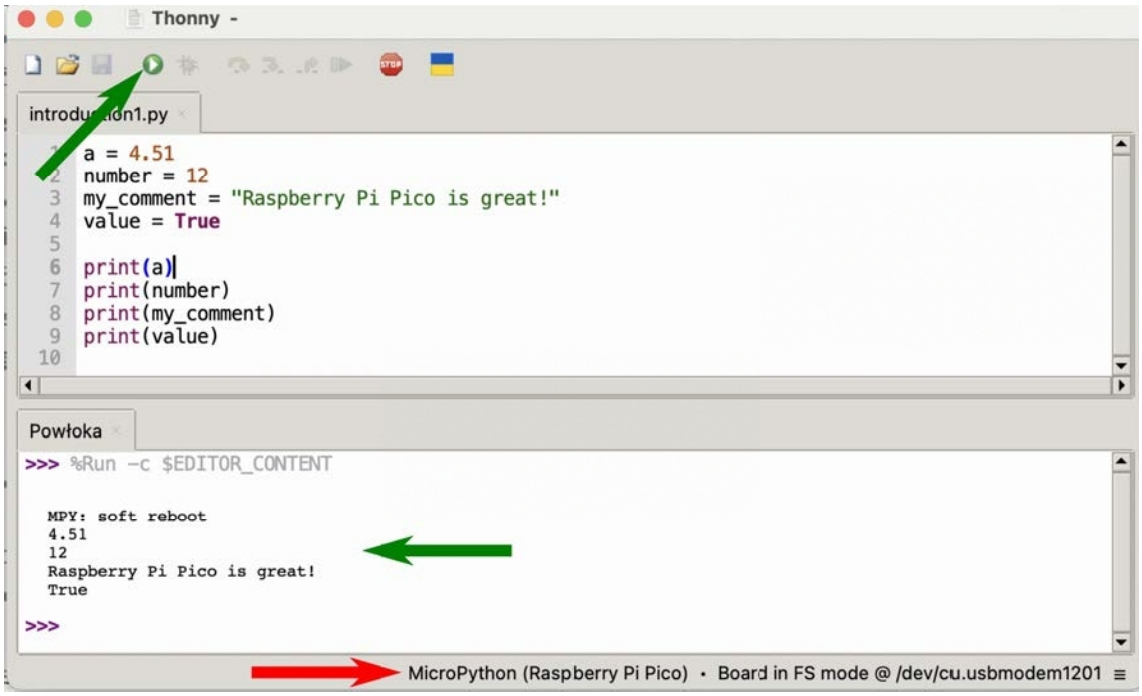
- **Variabele** - is een programmeerelement waarmee je een bepaalde waarde onder een zelfgekozen naam kunt opslaan, bijvoorbeeld als je een wiskundig programma maakt om de gebieden van
verschillende figuren, zouden we vaak het getal π gebruiken dat gelijk is aan, laten we aannemen: 3.14. In plaats van het handmatig in elke vergelijking in te voeren, kun je een variabele maken: pi=3.14, dan zou een voorbeeldprogramma om bijvoorbeeld de oppervlakte van een cirkel te tellen er als volgt uitzien:

```
pi=  3.14
r= 10
oppervlakte_cirkel= pi* r* r
```

Zie dat we hier drie numerieke variabelen hebben gemaakt. Eén die de waarde van het getal pi (pi) opslaat, de tweede die de waarde van de straal van de cirkel (r) opslaat en de derde die het resultaat opslaat, namelijk de oppervlakte van de cirkel (area_circle). Onthoud dat in Python en MicroPython variabelen worden gemaakt door eerst de variabelenaam van je keuze in te voeren en na het teken = de waarde. Er zijn vele soorten variabelen. In het bovenstaande voorbeeld hebben we variabelen van het *dubbele* type weergegeven (drijvendekommagetallen, bijvoorbeeld 3,42, -5,68, enzovoort). Daarnaast zijn er de basistypen variabelen:

- *geheel getal* (bijvoorbeeld 0, -4, 12, enz.),
- *boolean* (logische variabelen met twee mogelijke waarden: *True* of *False*),
- *string* (tekenreeksen, bijvoorbeeld "Hallo", "Raspberry Pi Pico").

Fig. 3.1 toont een voorbeeldprogramma waarin meerdere variabelen van verschillende typen zijn gemaakt. Om ze op het scherm weer te geven, kan de functie *print()* worden gebruikt. Wat we als argument van de printfunctie invoeren, wordt weergegeven in de terminal. Merk op dat je op de knop *Uitvoeren* (groene knop met een pijl) moet drukken om het programma te starten. Als deze grijs wordt weergegeven, controleer dan of het Raspberry Pi Pico-bord is aangesloten op de computer via USB en selecteer opnieuw: *Micropython (Raspberry Pi Pico)...* in de rechterbenedenhoek, zoals de pijl in de afbeelding laat zien. Als je het programma opslaat op de Raspberry Pi Pico met de naam "main.py" zal het automatisch starten zodra het Raspberry Pi Pico-bord wordt gevoed, ongeacht of het is aangesloten op de computer of wordt gevoed via bijvoorbeeld een Powerbank.



Afbeelding 3.1: Het voorbeeldprogramma met verschillende soorten variabelen.

- **Lijsten** - hiermee kun je een verzameling waarden maken. Een voorbeeldlijst ziet eruit als *values=[1, 35, 6, 14]*, wat betekent dat hij bestaat uit 4 elementen, gescheiden door komma's. De elementen zijn genummerd met indexen, net als huizen langs een straat, alleen met een verschil dat er geen punten zijn. De elementen zijn genummerd met indexen vergelijkbaar met huizen langs de straat, alleen met het verschil
  dat we ze nummeren vanaf nul en niet vanaf één. Dus om de waarde 6 uit de lijst te halen, moet je het commando: *values[2]* aanroepen, want het 0e element is waarde 1, het 1e element is waarde 35, het 2e element is waarde 6 en het 3e element is waarde 14. De indexen van de elementen staan tussen vierkante haakjes naast de naam van de lijst.
- **Functies** - zijn een subroutine, of een afzonderlijk deel van een programma dat een specifieke werking. Een functie kan invoergegevens accepteren die functieargumenten worden genoemd en kan een resultaat teruggeven. Je kunt bijvoorbeeld een functie schrijven die de oppervlakte van een cirkel berekent. Om een functie te declareren, gebruik je het woord *def* en dan de functienaam,
  bijvoorbeeld *circle_area*. Na de functienaam plaatsen we haakjes en binnenin geven we de functieargumenten. In het geval van het berekenen van de oppervlakte van een cirkel, hebben we de gebruiker nodig om de straal van de cirkel op te geven en dit zal ons functieargument zijn. Na de haakjes plaatsen we een dubbele punt. Binnen de functie plaatsen we de code die moet worden uitgevoerd wanneer de functie wordt aangeroepen, dat wil zeggen het berekenen van de oppervlakte van een cirkel met behulp van de formule: $A = \pi r^2$. Om ervoor te zorgen dat de functie een resultaat teruggeeft, gebruiken we het woord *return* en daarachter de waarde die de functie moet teruggeven. De code zou er dus als uitzien:

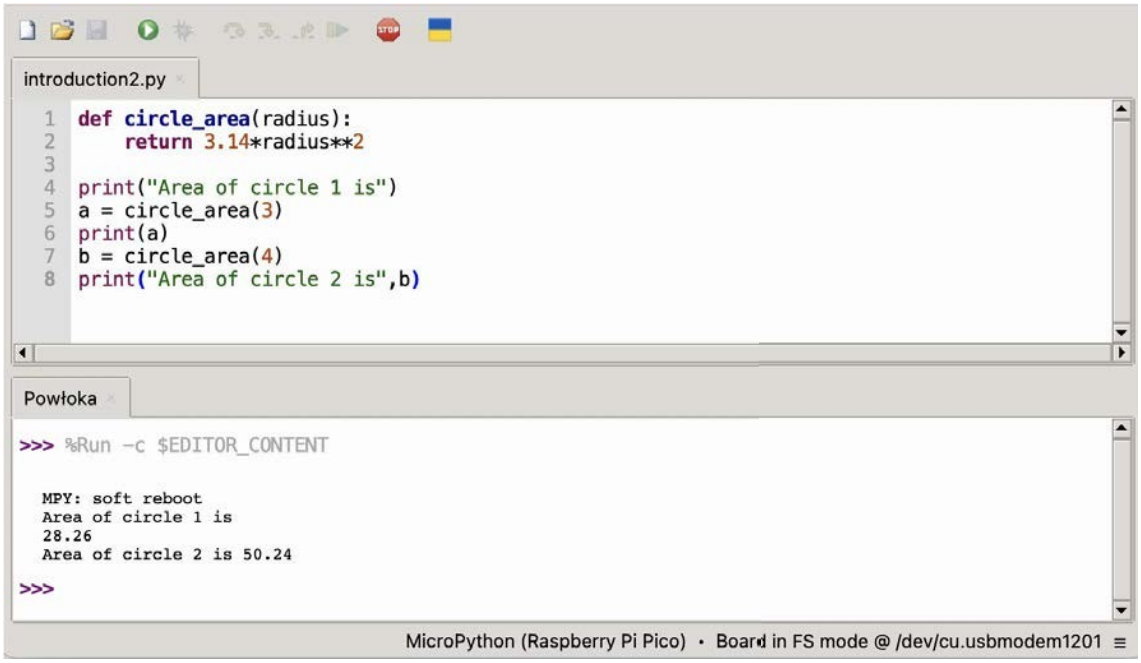
```
1 def circle_area (radius):
2     return 3.14* radius **2
```

Merk op dat het woord return 4 spaties na het woord def staat, dat op de regel erboven staat. In Micropython geeft inspringen aan welke regels binnen een functie/loop/structuur vallen en welke erbuiten. Het is erg belangrijk dat je inspringt, anders zal het programma de regels buiten de functie/loop/structuur uitvoeren.

tuur.

Het volgende punt is de exponentiatiebewerking. In MicroPython worden exponentatiebewerkingen uitgevoerd door twee sterren te plaatsen, dus x^y wordt geschreven als $x ** y$.

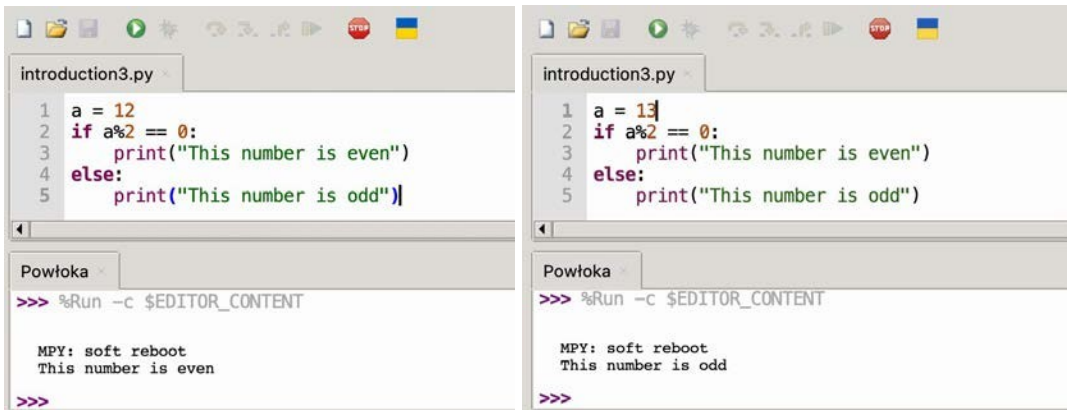
Om een bepaalde functie in een programma aan te roepen, hoef je alleen maar de naam op te geven en de argumenten tussen haakjes te zetten, zoals in Fig. 3.2. Merk op dat er twee alternatieve mogelijkheden worden getoond om de printfunctie aan te roepen. In het geval van het berekenen van de oppervlakte van de eerste cirkel, worden het commentaar en de waarde afzonderlijk weergegeven. Deze twee stukjes informatie staan dus op afzonderlijke regels na het aanroepen van het programma. Je kunt ook meerdere stukjes informatie tegelijk weergeven in de afdrukfunctie door ze te scheiden met komma's. Dan staat de weergegeven informatie op één regel. De weergegeven informatie staat dan op één regel.



Afbeelding 3.2: Het voorbeeldprogramma met de definitie van de functie.

- **if...**else-structuur - maakt het mogelijk om verschillende codefragmenten uit te voeren, afhankelijk van de voorwaarde waaraan wordt voldaan. Aan het begin schrijven we het woord *if* en daarachter de voorwaarde waaraan moet worden voldaan. Aan het einde plaatsen we een dubbele punt. Alle volgende
  Regels die we binnen *if* plaatsen (na inspringen) worden alleen uitgevoerd als aan de voorwaarde is voldaan. Dan kunnen we, maar dat hoeft niet, een else-blok toevoegen en daarachter een dubbele punt. Alle volgende regels binnen het *else* blok worden alleen uitgevoerd als niet aan de voorwaarde in de *if* is voldaan. Een voorbeeldprogramma in Fig. 3.3.
  Het voorbeeldprogramma gebruikt twee nieuwe elementen. Het eerste is de delingsoperatie met rest (%), die de rest geeft van de deling door een gegeven getal,
  Bijvoorbeeld 12%2 zal 0 opleveren omdat er geen rest is na het delen van 12 door 2, maar 13%2 zal 1 opleveren. Een ander nieuw ding is de gelijktekenoperatie. Als we willen controleren of een getal gelijk is aan een ander, gebruiken we het dubbele gelijkheidsteken (==). De overige vereffeningsbewerkingen zien er als volgt uit:
  - a>b (controleren of a groter is dan b)
  - a>=b (controleren of a groter is dan of gelijk aan b)

– a<b (controleren of a kleiner is dan b)
– a<=b (controleren of a kleiner is dan of gelijk aan b).

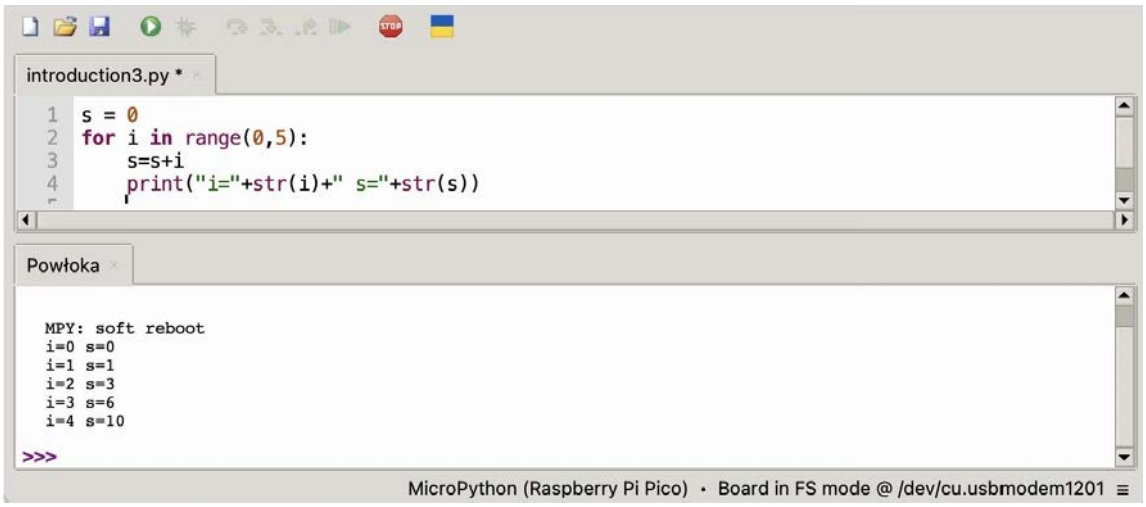


Afbeelding 3.3: Een voorbeeldcode die laat zien hoe je *de if...*else-structuur gebruikt.

In het bovenstaande programma kun je op een mooiere manier een bericht weergeven over of het getal even of oneven is. Met de printfunctie kun je verschillende tekenreeksen combineren door ze op te tellen met het "+"-teken. Om de variabele *a* weer te geven, die van het type geheel getal is, moet je deze eerst converteren naar een tekenreeks met de functie *str()*. Een voorbeeldprogramma wordt getoond in Fig. 3.4

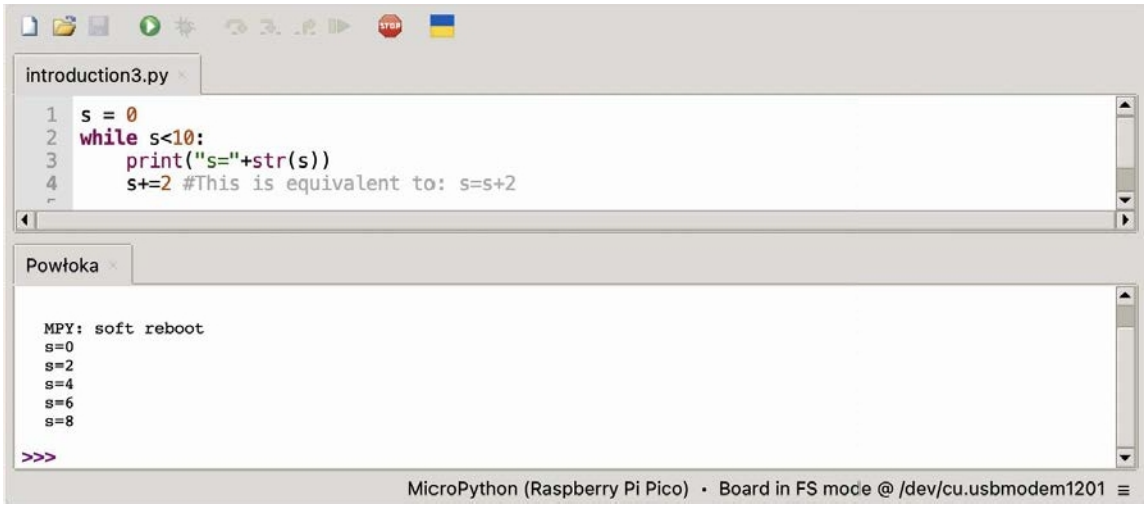
Afbeelding 3.4: Voorbeeldprogramma dat het aaneenschakelen van tekenreeksen toont om ze in één bericht weer te geven met de printfunctie.

- **For-lus** - is een lus waarmee je een bepaald stuk code meerdere keren kunt herhalen. Een voorbeeldprogramma met een for-lus wordt getoond in Fig. 3.5.
  Aan het begin staat het woord *for*, dan de itererende variabele (in dit geval *i* genoemd), dan het woord *in* en dan het bereik. In dit geval was het bereik numeriek van 0 tot 5, wat betekent dat de itererende variabele *i* elke keer dat de for-lus wordt uitgevoerd, zijn waarde met 1 zal verhogen, beginnend bij nul, tot een waarde die één minder is dan het bovenste bereik, dus tot en met 4. In dit geval zal de for-lus 5 keer worden uitgevoerd. In dit geval zal de for-lus 5 keer worden uitgevoerd en in elke iteratie zal de waarde van de iterator *i* worden toegevoegd aan de variabele *s*.



Afbeelding 3.5: Voorbeeldprogramma met for-lus.

- **While-lus** - voert een bepaald stuk code uit zolang aan de voorwaarde wordt voldaan. Een voorbeeldprogramma dat de werking van de while-lus laat zien, staat in Fig. 3.6. Aan het begin het woord *while*, dan de voorwaarde waaraan moet worden voldaan om de lus uit te voeren en helemaal aan het einde een dubbele . In dit geval de lus uitgevoerd totdat de variabele *s* minder dan 10 is. In het midden van de lus wordt de waarde van de variabele *s* weergegeven en vervolgens wordt de waarde verhoogd met 2. Deze bewerking kan op verschillende manieren worden geschreven. Een ervan is: *s*= *s*+ 2 of, in het kort, *s*+= 2, wat ook de waarde 2 zal toevoegen aan de huidige waarde van de variabele *s*.



Afbeelding 3.6: Voorbeeldprogramma met while-lus.

Het voorbeeldprogramma gebruikt het # teken. Alles wat op dezelfde regel na het # teken wordt geplaatst, is commentaar dat niet door het programma wordt uitgevoerd.

In het geval van microcontrollerprogrammering wordt meestal een oneindige lus gebruikt, die een bepaald fragment code steeds herhaalt totdat de voeding van het microcontrollerbord wordt . Meestal wordt een oneindige lus uitgevoerd met een while-lus

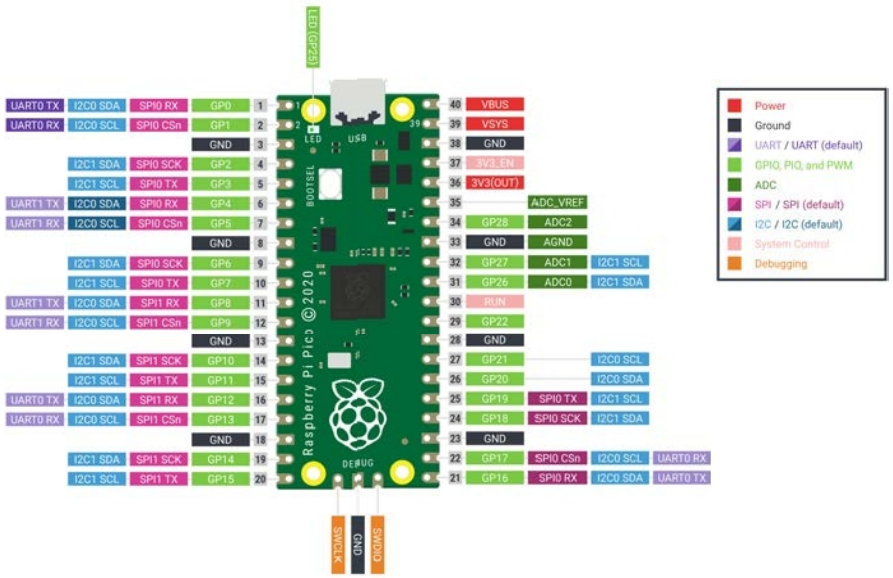
door een logische waarde van *True* op te geven als voorwaarde:

```
terwijl True:
    print(" Het is oneindige lus")
```

De geïntroduceerde, geselecteerde elementen van de MicroPython-taal zijn volgens de auteurs het meest nuttig om aan de slag te gaan met het Raspberry Pi Pico W-bord. Als je meer informatie nodig hebt over de MicroPython-taal, kun je het beste de documentatie over deze taal raadplegen.

4. De digitale signalen

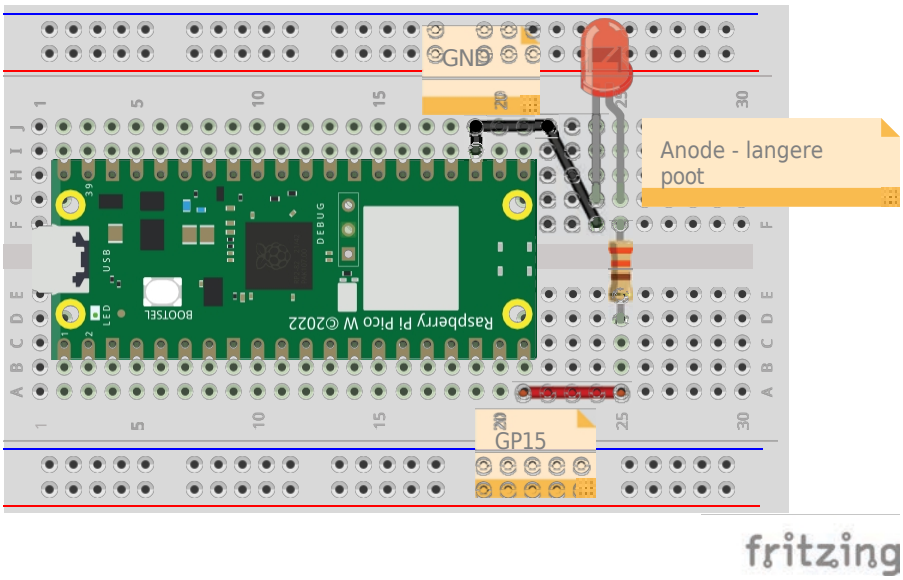
Digitale signalen zijn signalen die een van de twee mogelijke waarden 0 of 1 hebben, wat in het geval van de Raspberry Pi Pico microcontroller overeenkomt met de spanningswaarden 0 V of 3,3 V. Ze kunnen worden gebruikt om elementen zoals LED's te besturen (aan of uit) of om informatie te verkrijgen van elementen zoals een knop (ingedrukt of niet ingedrukt) of een bewegingssensor (beweging gedetecteerd of geen beweging gedetecteerd). Het lezen of genereren van digitale signalen gebeurt met behulp van GPIO-pinnen ***general-purpose input/output)***. GPIO zijn pinnen die zowel als ingangspennen kunnen worden gebruikt, waarvan we informatie kunnen lezen, bijvoorbeeld over het indrukken van een knop, of als uitgangspennen, waarop we digitale signalen kunnen genereren, bijvoorbeeld om een LED aan te zetten. Fig. 4.1 toont de 40 pinnen van het Raspberry Pi Pico-bord, die vanaf linksboven genummerd zijn. Pinnen die dienen als GPIO worden afgekort tot *GPx* (groene rechthoeken), waarbij *x* het rangtelwoord van de GPIO-pin is, beginnend bij 0 tot 28.



Afbeelding 4.1: De pinout van Raspberry Pi Pico. Bron van afbeelding: https://www.raspberrypi.com.

4.1 Voorbeeld 1: knipperende LED project

Laten we het Raspberry Pi Pico avontuur beginnen met een knipperend LED project. Sluit eerst de LED aan op de geselecteerde GPIO pin zoals getoond in Fig. 4.2.



Afbeelding 4.2: Het aansluiten van het elektronische circuit uit voorbeeld 1.

Laten we nu Thonny gebruiken om een programma in MicroPython te schrijven dat de diode elke 1s afwisselend aan en uit zet. Volg deze stappen om dit te doen:

1. Laten we beginnen met het opnemen van de bibliotheek in MicroPython:

```
import machine
```

De machinebibliotheek bevat alle nodige instructies om te communiceren met de Raspberry Pi Pico. Het commando *import* voegt de gespecificeerde bibliotheek toe aan het project.

2. Vervolgens moet je de GP15 pin configureren om als een output pin te werken, omdat we de LED besturen door een waarde van 1 of 0 naar de GP15 pin te sturen. Hiervoor wordt de Pin-functie uit de machinebibliotheek gebruikt. Om functies uit een bibliotheek aan te roepen in de MicroPython-taal, geven we eerst de naam van de bibliotheek op en na de punt de naam van de functie uit deze bibliotheek, dus *machine.Pin()*. De Pin-functie neemt twee argumenten. Het eerste is het GPIO-pinnummer. Het tweede is de specificatie of de GPIO-pin werkt als ingang (***machine.Pin.IN***) of als uitgang (***machine.Pin.OUT***). In dit geval zou de functieaanroep er dus uitzien als *machine.Pin(15, machine.Pin.OUT)*. Het object dat wordt teruggegeven door de Pin-functie wordt toegewezen aan de aangemaakte variabele *led*.

```
# Inclusief machinebibliotheek
Importeer machine

# GPIO configureren pin 15 als uitgang
led= machine. Pin (15 , machine. Pin. OUT )
```

3. In de volgende stap plaatsen we een oneindige lus, die het hoofdgedeelte van het programma zal bevatten:

```
# Inclusief machinebibliotheek
Importeer machine

# GPIO configureren pin 15 als uitgang
led= machine. Pin (15 , machine. Pin. OUT )

terwijl True:
```

4. In de volgende stap laten we de LED oplichten door de waarde 1 naar de GP15 pin te sturen. De functie *value()* wordt gebruikt om de waarde op de pin in te stellen. Deze neemt de waarde 0 of 1 als argument.

```
# Inclusief machinebibliotheek
Importeer machine

# GPIO configureren pin 15 als uitgang
led= machine. Pin (15 , machine. Pin. OUT )

terwijl True:
    led. waarde (1)
```

5. Nu moet het programma 1s wachten zodat we het effect van het oplichten van de diode kunnen zien. Hiervoor gebruiken we de utime-bibliotheek, die functies voor vertragingen bevat. Eerst moeten we de bibliotheek toevoegen:

```
# Inclusief bibliotheken
Importeer machine
importeer

# GPIO configureren pin 15 als uitgang
led= machine. Pin (15 , machine. Pin. OUT )

terwijl True:
    led. waarde (1)
```

6. Met de slaapfunctie van de utime-bibliotheek kun je het programma een aantal seconden vertragen. Laten we een vertraging van 1s instellen nadat de diode oplicht:

```
# Inclusief bibliotheken
Importeer machine
importeer utime

# GPIO configureren pin 15 als uitgang
led= machine. Pin (15 , machine. Pin. OUT )

terwijl True:
    led. waarde (1)
    utime . sleep (1)
```

7. De laatste stap is het uitschakelen van de LED door waarde 0 naar pin GP15 te sturen en 1s te wachten om het effect te zien.

```
# Inclusief bibliotheken
Importeer machine
importeer utime

# GPIO configureren pin 15 als uitgang
led= machine. Pin (15 , machine. Pin. OUT )

terwijl True:
    led. waarde (1)
    utime . sleep (1)
    led. waarde (0)
    utime . sleep (1)

```

Voer de bovenstaande code uit in de Thonny editor en bekijk het effect. Als de diode elke 1s knippert, heb je alles goed gedaan. Zo niet, controleer dan eerst de aansluiting van de LED-diode op de printplaat en controleer daarna of je geen fout hebt gemaakt in het programma bij het herschrijven van de code.

Tip 4.1.1

In het bovenstaande voorbeeld stellen we de waarde van de GP15-pin in op 1 of 0 om de diode in of uit te schakelen. Hiervoor is de functie value() gebruikt. Het programma kan echter nog eenvoudiger worden geschreven. De toggle() functie verandert de waarde op een gegeven pin in het tegenovergestelde, d.w.z. als de waarde was ingesteld op 1, verandert deze in 0, en als de waarde was ingesteld op 0, verandert deze in 1. Dus in het bovenstaande voorbeeld zou het binnenste van de oneindige lus veranderen in:

```
terwijl True:
    led. toggle ()
    utime . sleep (1)
```

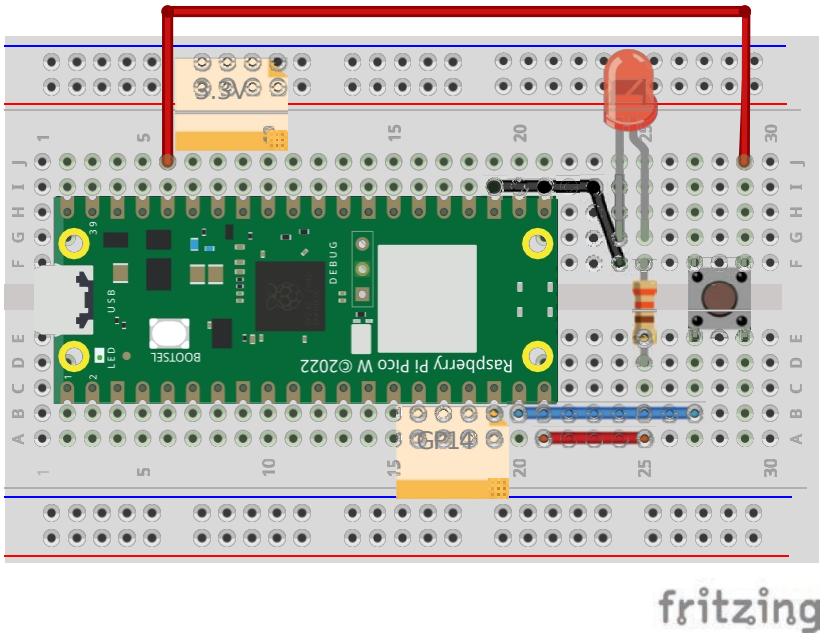
Tip 4.1.2

Als je het programma opslaat op de Raspberry Pi Pico (niet op de computer) met de naam "main.py" zal het automatisch starten wanneer het Raspberry Pi Pico bord wordt gevoed, ongeacht of het is aangesloten op de computer of wordt gevoed via bijvoorbeeld een Powerbank.

4.2 Voorbeeld 2: LED aan/uit met drukknop

In dit voorbeeld gaat de LED aan of uit wanneer de knop wordt ingedrukt. Sluit hiervoor eerst de schakeling aan volgens Fig. 4.3.

Drukknop is aangesloten op GP14 pin en LED op GP15. Open nu Thonny en laten we wat MicroPython-code schrijven die de LED aanstuurt, afhankelijk van het indrukken van de knop. Volg hiervoor de onderstaande stappen:



Figuur 4.3: Het aansluiten van het elektronische circuit uit voorbeeld 2.

1. Eerst moet je de machinebibliotheek toevoegen en de GP15-pin, waarop de LED is aangesloten, configureren als een uitgang, zoals in het vorige voorbeeld.

```
import machine

led= machine. Pin (15 , machine. Pin. OUT )
```

2. Vervolgens moet je de GP14-pin, waarop de knop is aangesloten, configureren als een ingangselement (optie **machine.Pin.IN**) omdat we de waarde van de knop zullen lezen (of hij is ingedrukt). De knop moet ook worden verbonden met weerstanden genaamd *pull-up* of *pull-down*, die al zijn gemonteerd in het Raspberry Pi Pico-bord en verbonden met alle GPIO-pinnen. Deze weerstanden zijn nodig om zwevende input te voorkomen, d.w.z. een situatie waarin informatie van de GPIO-pin wordt gelezen dat de knop is ingedrukt terwijl we deze niet hebben ingedrukt. Dit kan worden beschouwd als een storing en een ernstige ontwerpfout. Pull-down-weerstanden verbinden de knop met massa, wat betekent dat als de knop niet wordt ingedrukt, de waarde 0 wordt gelezen. Pull-up weerstanden verbinden de knop met 3,3V, wat betekent dat als de knop niet wordt ingedrukt, de waarde 1 wordt gelezen. In dit geval gebruiken we pull-downweerstanden. Hiervoor moet **machine.Pin.PULL_DOWN** worden opgegeven als derde argument van de Pin-functie en moet de tegenoverliggende knopdraad worden verbonden met +3,3 V. Als we pull-upweerstanden willen gebruiken, moeten we **machine.Pin.PULL_UP** opgeven als derde argument en moet de tegenoverliggende knopdraad worden verbonden met aarde (GND), die een potentiaal van 0 V heeft.

```
import machine

led = machine. Pin (15 , machine. Pin. OUT ) but=
machine . Pin (14 , machine . Pin. IN , machine . Pin. PULL_DOWN )
```

3. Laten we in de volgende stap eerst de LED uitschakelen door de waarde 0 in te stellen:

```
import machine

led = machine.Pin(15, machine.Pin.OUT) but=
machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_DOWN)

led.waarde(0)
```

4. Laten we nu de hoofdlus van het programma maken daarin de toestand van de knop controleren. Als de waarde die de knop teruggeeft gelijk is aan 1, betekent dit dat de knop werd ingedrukt:

```
import machine

led = machine.Pin(15, machine.Pin.OUT) but=
machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_DOWN)

led.waarde(0)

terwijl True:
    als maar.waarde()==1:
```

5. Laten we vervolgens de LED-status veranderen in het tegenovergestelde wanneer de knop wordt :

```
import machine

led = machine.Pin(15, machine.Pin.OUT) but=
machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_DOWN)

led.waarde(0)

terwijl True:
    als maar.waarde()==1:
        led.toggle()
```

6. De laatste stap is het toevoegen van een vertraging van 1s. Deze vertraging kan korter zijn, maar is noodzakelijk omdat wanneer we op de knop drukken deze enkele milliseconden aan is voordat we de knop loslaten. Gedurende deze tijd zal het programma de hoofdlus meerdere keren uitvoeren en zal het commando om de LED-status te veranderen naar het tegenovergestelde meerdere keren worden uitgevoerd. Voeg een vertraging toe om dit te voorkomen. Vergeet niet de utime-bibliotheek toe te voegen:

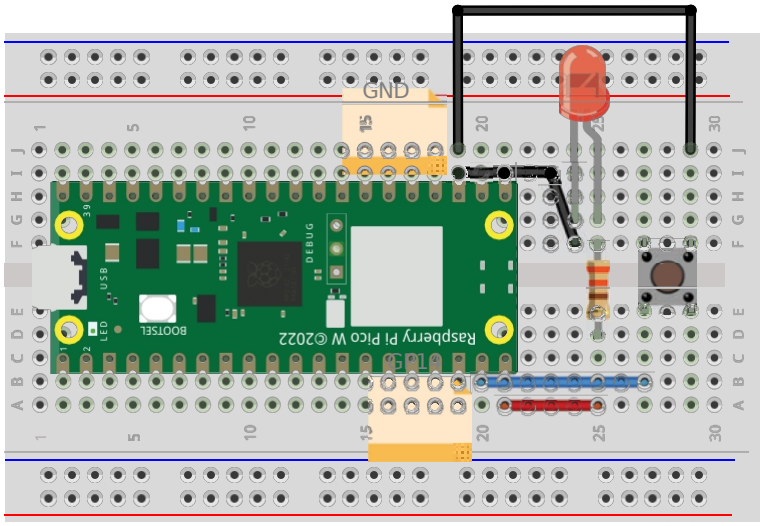
```
importeer machine
importeer utime

led = machine.Pin(15, machine.Pin.OUT) but=
machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_DOWN)

led.waarde(0)

terwijl True:
    als maar.waarde()==1:
        led.toggle()
```

```
12              utime. slaap (1)
```

Voer de bovenstaande code uit in de Thonny editor en druk op de knop die is aangesloten op de Raspberry Pi. Gaat de LED branden? Zo niet, controleer dan de aansluiting van de knop.

Waarschuwing 4.2.1

Als je Raspberry Pi Pico 2W gebruikt, heeft de RP2350 microcontroller een probleem met het verlagen van de spanning naar 0V bij gebruik van pull-down weerstanden. In werkelijkheid daalt de spanning tot ongeveer 2,2V. Om deze reden is het beter om pull-up weerstanden te gebruiken die de knop verbinden met 3,3V, wat betekent dat als de knop niet wordt ingedrukt, de waarde 1 wordt gelezen. Om dit te doen, sluit je de poot van de knop aan op 0V in plaats van 3,3V zoals weergegeven in de figuur:

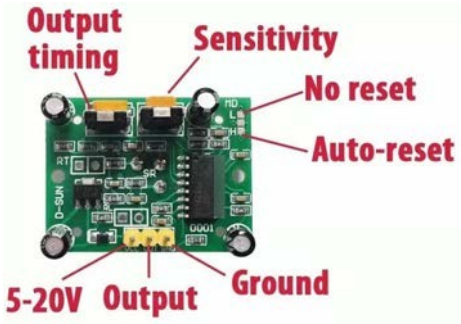


Er moeten echter twee wijzigingen worden aangebracht in het programma. De eerste is het veranderen van de constante *machine.Pin.PULL_DOWN* in *machine.Pin.PULL_UP* bij het configureren van de knop. De tweede wijziging is dat de knop wordt ingedrukt als we de waarde 0 en niet 1 lezen op de pin waarop de knop is aangesloten:

```
1  importeer machine
2  importeer utime
3
4  led     =      machine. Pin (15 ,      machine. Pin . OUT ) but=
5  machine . Pin (14 , machine . Pin . IN , machine . Pin . PULL_UP )
6
7  led. waarde (0)
8
9  terwijl True:
10       als   maar. waarde () ==0:
11            led. toggle () utime
12       . sleep (1)
```

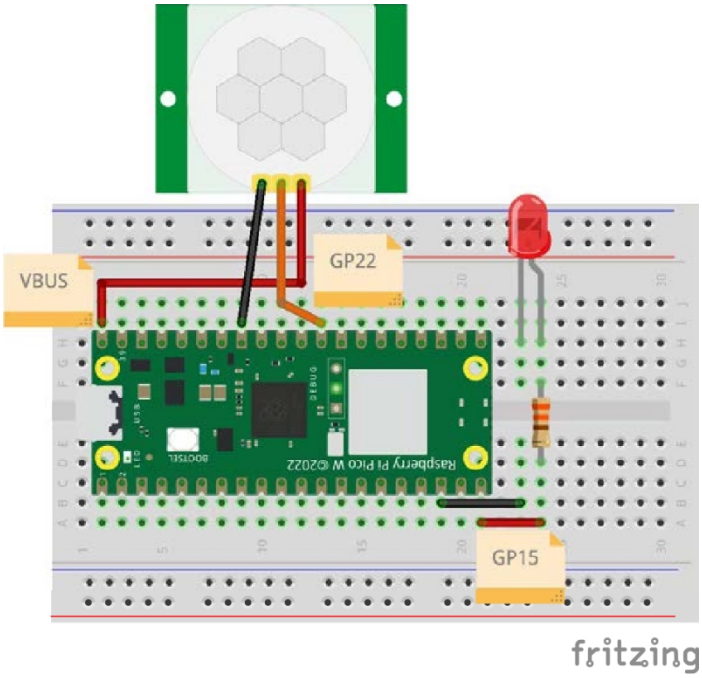
4.3 Voorbeeld 3: Licht ingeschakeld door een bewegingssensor

In dit voorbeeld maken we een automatische lichtschakelaar die het licht inschakelt wanneer beweging wordt gedetecteerd door de PIR bewegingssensor. Hiervoor gebruiken we een LED diode en een HC-SR501 PIR bewegingssensor uit Fig. 4.4 Volgens de documentatie van de HC-SR501 PIR sensor moet deze sensor worden aangesloten op een voeding van 5V tot 20V. Op het Raspberry Pi Pico W bord wordt een 5V voeding geleverd op de *VBUS* pin. Als je sensoren gebruikt die gevoed worden door een hogere spanning dan 3,3V, zorg er dan voor dat het teruggezonden signaal een spanning heeft van maximaal 3,3V, anders kunnen we het Raspberry Pi Pico-bord beschadigen. Waar kan ik dergelijke informatie vinden? De eenvoudigste manier is om een datasheet voor een bepaalde sensor op internet te vinden en te lezen welke spanning het hoge signaal heeft dat door de sensor wordt gegenereerd. In dit geval geeft de fabrikant aan dat de HC-SR501 PIR-sensor signalen teruggeeft met een waarde van 3,3V of 0V.



Afbeelding 4.4: PIR-bewegingssensor HC-SR501. Bron: https://www.unoelectro.com.ar.

Sluit dus de PIR bewegingssensor en LED diode aan op het Raspberry Pi Pico W bord zoals getoond in Fig. 4.5.



Afbeelding 4.5: Het aansluiten van het elektronische circuit uit voorbeeld 3.

Laten we nu een programma schrijven in de Thonny editor dat de LED 5 seconden laat branden wanneer beweging wordt gedetecteerd. Volg hiervoor deze stappen:

1. Eerst moet je de *machine* en *utime* bibliotheken importeren:

```
importeer machine
importeer utime
```

2. Vervolgens moet je de GP15-pin waarop de LED is aangesloten configureren als uitgang en de LED uitschakelen:

```
importeer machine
importeer utime

LED= machine. Pin (15 , machine. Pin. OUT ) LED
. waarde (0)
```

3. De PIR-bewegingssensor is een ingangselement dat 0 of 1 teruggeeft, afhankelijk van of het beweging heeft gedetecteerd. Daarom moet de GP22 pin, waarop de PIR bewegingssensor is aangesloten, worden geconfigureerd als een ingang:

```
importeer machine
importeer utime

LED= machine. Pin (15 , machine. Pin. OUT ) LED
. waarde (0)

PIR= machine. Pin (22 , machine. Pin. IN)
```

4. Nu moet je in de hoofdlus de waarde lezen die de bewegingssensor teruggeeft en deze weergeven in de terminal:

```
importeer machine
importeer utime

LED= machine. Pin (15 , machine. Pin. OUT ) LED
. waarde (0)

PIR= machine. Pin (22 , machine. Pin. IN) while

True:
    beweging= PIR . waarde ()
    print( beweging )
```

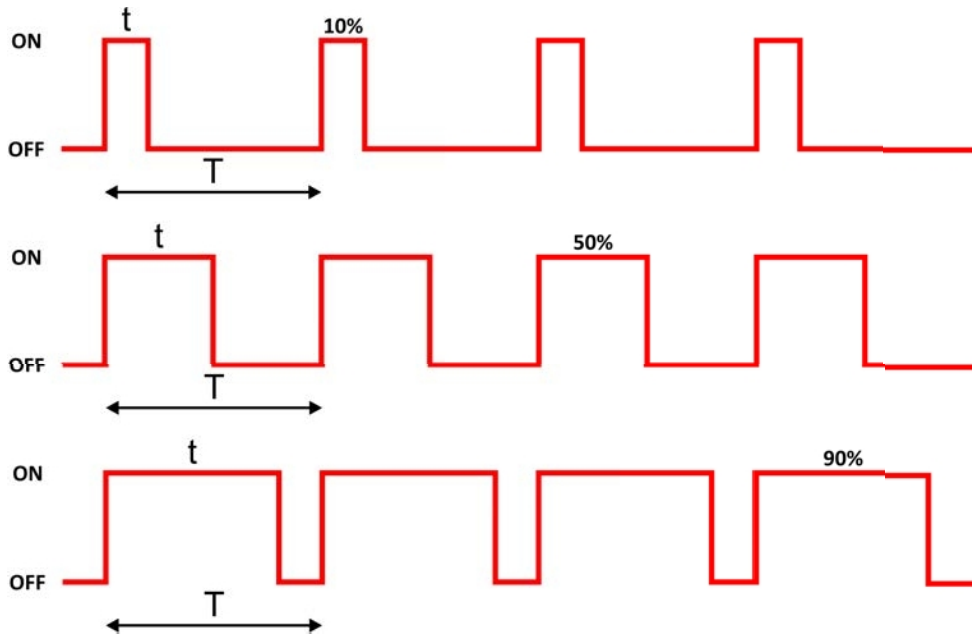
5. Start nu het programma en controleer hoe de bewegingssensor zich gedraagt. Fig. 4.4 toont twee potentiometers gemarkeerd als "Uitgangstijdstip" en "Gevoeligheid". Je kunt ze gebruiken om de bewegingssensor aan te passen. We raden aan om de potentiometer "Uitgangstijdstip" op de laagst mogelijke waarde in te stellen en de gevoeligheid naar eigen smaak aan te passen. Maak je geen zorgen als de sensor na het detecteren van beweging meerdere keren de waarde 1 teruggeeft. Dit is normaal omdat de kortste signaalduur langer is dan de uitvoeringstijd van de hoofdprogrammalus. Zodra je de bewegingssensor hebt aangepast, ga je verder met het schrijven van het programma.
6. In de volgende stap moet de LED oplichten als er gedurende 5 seconden beweging wordt gedetecteerd, anders moet de LED uit zijn:

```
importeer machine
importeer utime

LED= machine. Pin (15 , machine. Pin. OUT ) LED
. waarde (0)

PIR= machine. Pin (22 , machine. Pin. IN) while

True:
    beweging= PIR . waarde ()
    print( motion )
    if( motion ==1) :
        LED . waarde (1)
        utime. sleep (5)
    anders:
        LED . waarde (0)
```

Het programma is klaar. Test de werking.

4.4 Pulsbreedtemodulatie (PWM)

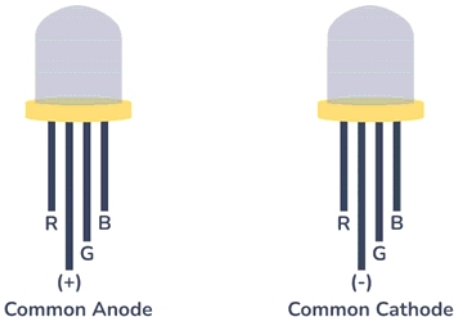
Sommige componenten zoals RGB-LED's of servo's worden aangestuurd met behulp van Pulse Width Modu- lation (PWM). Met deze techniek kunnen rechthoekige signalen worden gegenereerd met een gespecificeerde duty cycle van 0 tot 100%. Als de duty cycle bijvoorbeeld is ingesteld op 20%, dan hebben we voor 20% van de pulsduur een hoog signaal en voor 80% van de pulsduur een laag signaal. Voorbeeldsignaalvormen gegenereerd met de PWM-techniek worden getoond in Fig. 4.6.



Afbeelding 4.6: Voorbeeld PWM-signalen met duty cycles van 10% (bovenste plot), 50% (middelste plot) en 90% (onderste plot). Bron: https://www.robotyka.net.pl/pwm-modulacja-sz erokosci-impulsu/.

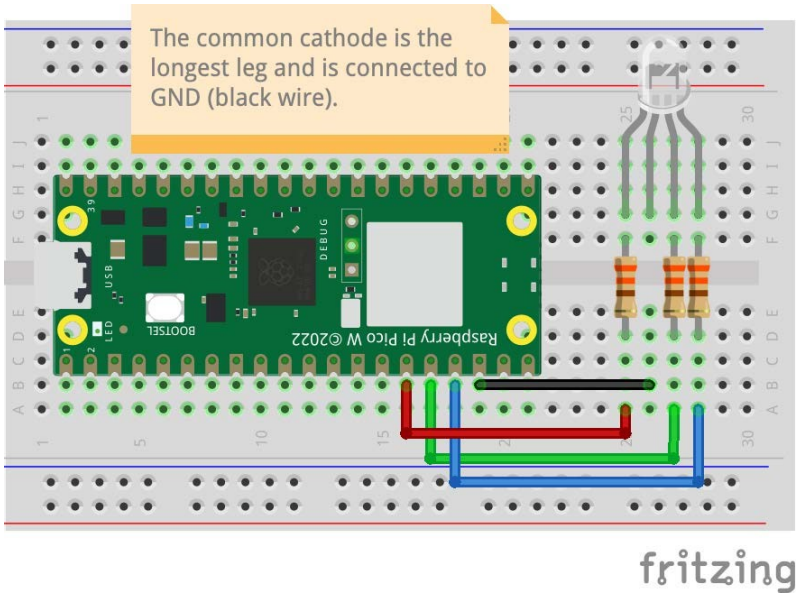
4.4.0.1 Voorbeeld 4: RGB LED

Een RGB-diode bestaat uit drie diodes in de kleuren rood, groen en blauw. Om een geselecteerde kleur te genereren, regel je de vulling van elke kleur met een PWM. Olijf is bijvoorbeeld een mengsel van rood en groen. Om olijf te genereren, moet je in de RGB-code 50% rode vulling en 50% groene vulling instellen. PWM-signalen maken dit mogelijk en daarom worden ze gebruikt om RGB-diodes aan te sturen. De RGB LED heeft 4 pinnen zoals weergegeven in Fig. 4.7.



Afbeelding 4.7: Soorten RGB LED-diodes. Bron: https://www.build-electronic-circuits.com/rgb-led/ .

De langste draad is de gemeenschappelijke draad. Als we een RGB-diode met gemeenschappelijke kathode gebruiken, moet de langste draad worden verbonden met GND. En het verlichten van een specifieke kleur gebeurt door de hoge stand in te stellen op een rode, groene of blauwe draad. Als de RGB-diode een gemeenschappelijke anode heeft, moet de langste draad worden verbonden met 3,3 V en wordt een specifieke kleur verlicht door de lage stand in te stellen op een bepaalde draad. In dit voorbeeld gebruiken we een RGB-diode met een gemeenschappelijke kathode, die moet worden aangesloten zoals weergegeven in Fig. 4.8. De resterende 3 pootjes zijn de aansluitdraden voor de afzonderlijke kleuren: rood, groen en blauw. Vergeet niet om stroombegrenzende weerstanden toe te voegen aan elke kleurbesturingspen (3 x 330 Ω zou optimaal moeten zijn).



Afbeelding 4.8: Voorbeeldaansluiting van een RGB-LED met gemeenschappelijke kathode. We gebruikten 3 x 330 Ω
weerstanden voor stroombegrenzing van elke LED.

Nadat je de RGB-diode hebt aangesloten, open je Thonny en begin je met het schrijven van de code die de kleurreeksen zal weergeven: rood, groen, blauw, roze, wit en geel. Om dit te doen, moet je

1. Voeg eerst de *machine*- en utime-bibliotheken toe:

```
importeer machine
importeer utime
```

2. Vervolgens moet je definiëren welke pinnen worden gebruikt om PWM-signalen te genereren. Dit wordt gedaan met de *PWM* functie uit *de machine* bibliotheek, die een *Pin* object als argument neemt. Zoals gezegd bestaat de RGB LED uit drie diodes (rood, groen en blauw), dus je moet drie pinnen configureren als PWM.

```
importeer machine
importeer utime

rode= machine. PWM ( machine . Pin (11))
groen= machine. PWM ( machine. Pin (12)) blauw=
machine. PWM ( machine. Pin (13))
```

3. Vervolgens moet je de frequentie van het PWM-signaal instellen. Daar is de freq-functie voor:

```
importeer machine
importeer utime

rode= machine. PWM ( machine . Pin (11))
groen= machine. PWM ( machine. Pin (12)) blauw=
machine. PWM ( machine. Pin (13))

rood. freq (1000)
groen. freq (1000)
blauw. freq (1000)
```

4. In Raspberry Pi Pico W is de resolutie van PWM-signalen 16 bits. Dit betekent dat de vulling van het PWM-signaal is ingesteld van 0 tot 65535, waarbij 65535 100% vulling is. Kleuren worden echter gegeven in RGB-code, waarbij de waarden van 0 tot 255 lopen. Daarom is het handig om functies te maken voor het instellen van de vulling op een bepaalde diode, die de waarde in het bereik (0; 255) converteren naar het bereik (0; 65535) zodat de gebruiker deze waarden niet handmatig opnieuw hoeft te berekenen. Om waarden gegeven in het bereik (0; 255) te converteren naar het bereik (0; 65535), moeten de gegeven waarden worden vermenigvuldigd met 65535/255=257:

```
importeer machine
importeer utime

rode= machine. PWM ( machine . Pin (11))
groen= machine. PWM ( machine. Pin (12)) blauw=
machine. PWM ( machine. Pin (13))

rood. freq (1000)
groen. freq (1000)
blauw. freq (1000)


```

```
def set_color(r, g, b):
    rood. duty_u16 (r*257)
    groen. duty_u16 (g *257)
    blauw. duty_u16 (b
    *257)
```

5. Nu moeten we de hoofd while-lus maken, waarin de kleuren rood, groen en blauw worden ingesteld en er een vertraging van 1s tussen zit:

```
importeer machine
importeer utime

rode= machine. PWM ( machine . Pin (11))
groen= machine. PWM ( machine. Pin (12)) blauw=
machine. PWM ( machine. Pin (13))

rood. freq (1000)
groen. freq (1000)
blauw. freq (1000)

def set_color(r, g, b): rood.
    duty_u16 (r*257) groen.
    duty_u16 ( g *257) blauw.
    duty_u16 ( b *257)

terwijl True:
    set_color (255 ,0 ,0) # rood
    utime . sleep (1)

    set_color (0,255,0) # groen
    utime . sleep (1)

    set_color (0 ,0 ,255) # blauw
    utime . sleep (1)
```

6. Start het programma en kijk of deze drie kleuren in volgorde worden weergegeven op de RGB-diode. Zo niet, controleer dan de aansluiting en de code. Als ze worden weergegeven, voeg dan de resterende kleuren toe om weer te geven, d.w.z.: roze, wit en geel.

```
importeer machine
importeer utime

rode= machine. PWM ( machine . Pin (11))
groen= machine. PWM ( machine. Pin (12)) blauw=
machine. PWM ( machine. Pin (13))

rood. freq (1000)
groen. freq (1000)
blauw. freq (1000)

def set_color(r, g, b): rood.
    duty_u16 (r*257) groen
    . duty_u16 ( g *257)
```

```
    blauw. duty_u16 ( b     )
    *257

terwijl   True:
    set_color (255,0,0)      # rood
    utime . slaap (1)
    set_color (0,255,0)      # groen
    utime . slaap (1)
    set_color (0,0,255)      # blauw
    utime . slaap (1)
    set_color (255,20 ,147) # roze
    utime .      sleep    (1) set_color
    (255 ,255 ,255) # wit utime . sleep
    (1)
    set_color (255 , 255 , 0) # geel utime .
    sleep (1)
```

Test het voltooide programma.

Tip 4.4.1

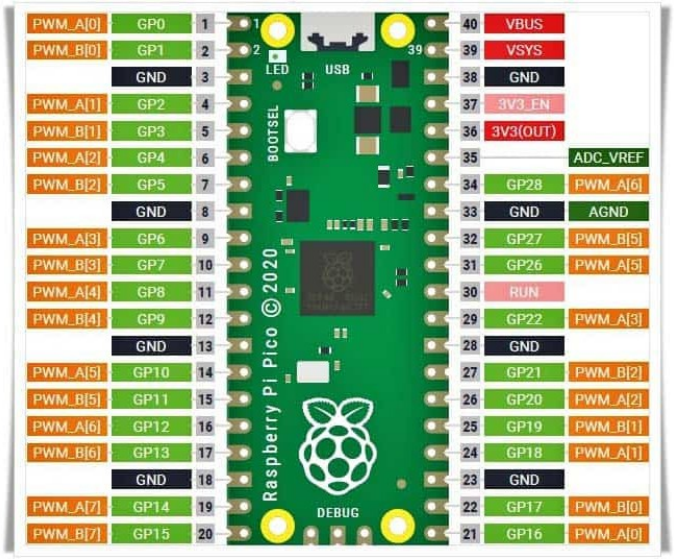
Wat moet er in het bovenstaande voorbeeld worden veranderd als we een RGB-LED met gemeenschappelijke anode hebben? Er moeten twee dingen worden veranderd:

1. De aansluiting van de RGB LED. De zwarte draad uit Fig. 4.8 moet worden verbonden met 3,3V en niet met GND.
2. De individuele kleuren in de gemeenschappelijke anode RGB LED worden verlicht met een lage toestand en niet met een hoge toestand. Dus om 100% vulling in te stellen, moet je de waarde 0 instellen en niet 65535 op een bepaalde pin. Daarom moet je de *set_color*() functie als volgt aanpassen:

```
def set_color(r, g, b):
    rood. duty_u16 (65535 - r*257)
    groen. duty_u16 (65535 - g *257)
    blauw. duty_u16 (65535 - b *257)
```

Waarschuwing 4.4.1

Wees voorzichtig als je meer pinnen gebruikt die geconfigureerd zijn om PWM-signalen te genereren. In Raspberry Pi Pico W hebben we 8 PWM-kanalen, elk met twee uitgangen A en B (zie Fig. 4.9). Elke uitgang met hetzelfde nummer, bijvoorbeeld A[5] en B[5] zijn niet volledig onafhankelijk. We kunnen deze twee uitgangen in één programma gebruiken en verschillende PWM-belastingscycli instellen, maar de frequentie van deze signalen zal hetzelfde zijn.



Afbeelding 4.9: De PWM-kanalen. Bron: https://www.codrey.com/raspberry-pi/raspberry-pi-pico-pwm-primer/.

5. De analoge signalen

Naast digitale signalen die twee mogelijke toestanden 0 of 1 hebben (0 V of 3,3 V overeenkomstig in de laagspannings-TTL-standaard), geven analoge signalen informatie in termen van spanning of stroom. Deze geven continue waarden in een bepaald bereik, bijvoorbeeld een sensor die temperatuur meet zal een spanning teruggeven die evenredig is met de temperatuur (bijvoorbeeld 235 mV komt overeen met een temperatuur van 23,5^oC.). We kunnen zeggen dat de versterking van de sensor gelijk is aan 10 $mV/^oC$). Om de spanning van analoge signalen af te lezen is een analoog-digitaalomzetter (ADC) nodig. Zo'n apparaat is al ingebouwd in de Raspberry Pi Pico microcontroller. De ingebouwde converter is 12-bits. Als we nauwkeurigere metingen willen, kunnen we een externe converter aansluiten op de Raspberry Pi Pico (meer hierover later in de handleiding).

5.1 Voorbeeld 5: temperatuurmeting

Temperatuur meten kan op twee manieren: met de ingebouwde temperatuursensor (biased bipolaire diode) of met een externe temperatuursensor, bijvoorbeeld LM35. In dit geval worden beide methoden geïmplementeerd en de resultaten vergeleken. Laten we beginnen met de ingebouwde temperatuursensor. Open de Thonny editor en voer de volgende stappen uit:

1. Eerst moet je de *machine* en *utime* bibliotheken toevoegen:

```
importeer machine
importeer utime
```

2. Vervolgens moet je de pin configureren om als ADC te werken, hiervoor wordt de *ADC(pin nummer)* functie gebruikt. De ingebouwde temperatuursensor wordt aangesloten op het vierde ADC-kanaal:

```
importeer machine
importeer utime

built_in_temp= machine. ADC (4)
```

3. Lees dan in de hoofd while-lus de waarde van de ADC met de functie *read_u16()*. Hoewel de ADC 12-bits is, zet de functie *read_u16* de waarde onmiddellijk om van 12bits naar 16bits:

```
importeer machine
importeer utime

built_in_temp= machine. ADC (4)

terwijl True:
    built_in_temp. read_u16 ()
```

4. Om de spanningswaarde te verkrijgen, converteer je de waarden die worden geretourneerd door de functie *read_u16* in het bereik (0;65535) naar de spanning (0;3,3)V:

```
importeer machine
importeer utime

built_in_temp= machine. ADC (4)

terwijl True:
    voltage1= built_in_temp . read_u16 () * 3 .3 /65535
```

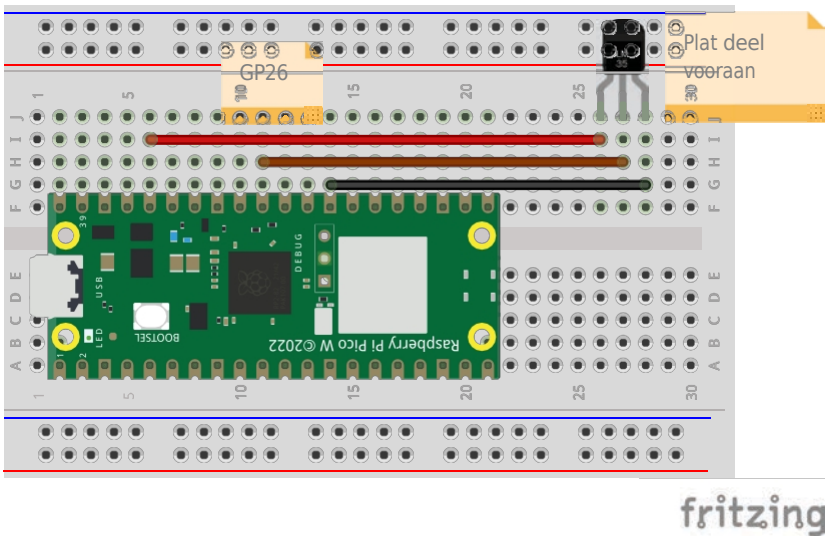
5. Vervolgens moet je de spanning omrekenen naar temperatuur volgens de formule in de documentatie van de RP2040. Het resultaat wordt elke 1s weergegeven op de terminal:

```
importeer machine
importeer utime

built_in_temp= machine. ADC (4)

terwijl True:
    voltage1= built_in_temp . read_u16 () * 3 .3 /65535
    temp1= 27 - ( voltage1 -0.706) /0.001721 print(
    temp1 )
    utime . slaap (1)
```

Test het programma.

Metingen met de ingebouwde temperatuursensor gaan gebukt onder twee fundamentele problemen. Het eerste is de hoge onnauwkeurigheid van de temperatuurmeting als gevolg van de gevoeligheid van de afgelezen waarden afhankelijk van de referentiespanning. Een verandering van de referentiespanning met 1% veroorzaakt al een temperatuurafleesfout van ongeveer 4^oC. Een ander probleem is het feit dat de ingebouwde temperatuursensor eigenlijk de temperatuur van de RP2040 meet en niet die van de omgeving. Daarom kan tijdens intensief gebruik van de RP2040 de afgelezen temperatuur veel hoger zijn dan de omgevingstemperatuur. Als je een nauwkeurigere temperatuurmeting wilt, kun je beter een externe temperatuursensor gebruiken.

Laten we nu het programma uitbreiden door temperatuurmetingen van de analoge LM35-sensor toe te voegen. Sluit hiervoor de LM35-sensor aan op de Raspberry Pi Pico W zoals getoond in Fig. 5.1. De pinnen waarop de ADC-kanalen worden uitgelood zijn: GP26, GP27 en GP28 (zie Fig. 1.1 - donkergroene markeringen).



Afbeelding 5.1: Voorbeeldaansluiting van een LM35-sensor.

6. Laten we nu de GP26 pin configureren om als ADC te werken:

```
importeer machine
importeer utime

built_in_temp= machine. ADC (4)
externe_temp= machine.ADC(26

terwijl True:
      voltage1= built_in_temp . read_u16 () * 3 .3 /65535
      temp1= 27 - ( voltage1 -0.706) /0.001721 print(
      temp1 )
      utime . slaap (1)
```

7. Lees in de volgende stap de spanning af die de LM35 sensor teruggeeft en vermenigvuldig het resultaat met 100 om de temperatuur in graden Celsius te krijgen volgens de documentatie van de LM35 sensor (sensorversterking is 10 $mV/^oC$, dus we hebben spanning in mV nodig, dus we vermenigvuldigen met 1000 en delen door sensorversterking: 10, om temperatuur in ^oC te krijgen. Vermenigvuldig de waarde na ADC-conversie dus met 100):

```
importeer machine
importeer utime

built_in_temp= machine. ADC (4) external_temp=
machine. ADC (26)

terwijl True:
          voltage1= built_in_temp . read_u16 () * 3 .3 /65535
          temp1= 27 - ( voltage1 -0.706) /0.001721

          voltage2= external_temp . read_u16 () * 3 .3 /65535
          temp2= voltage2 *100
```

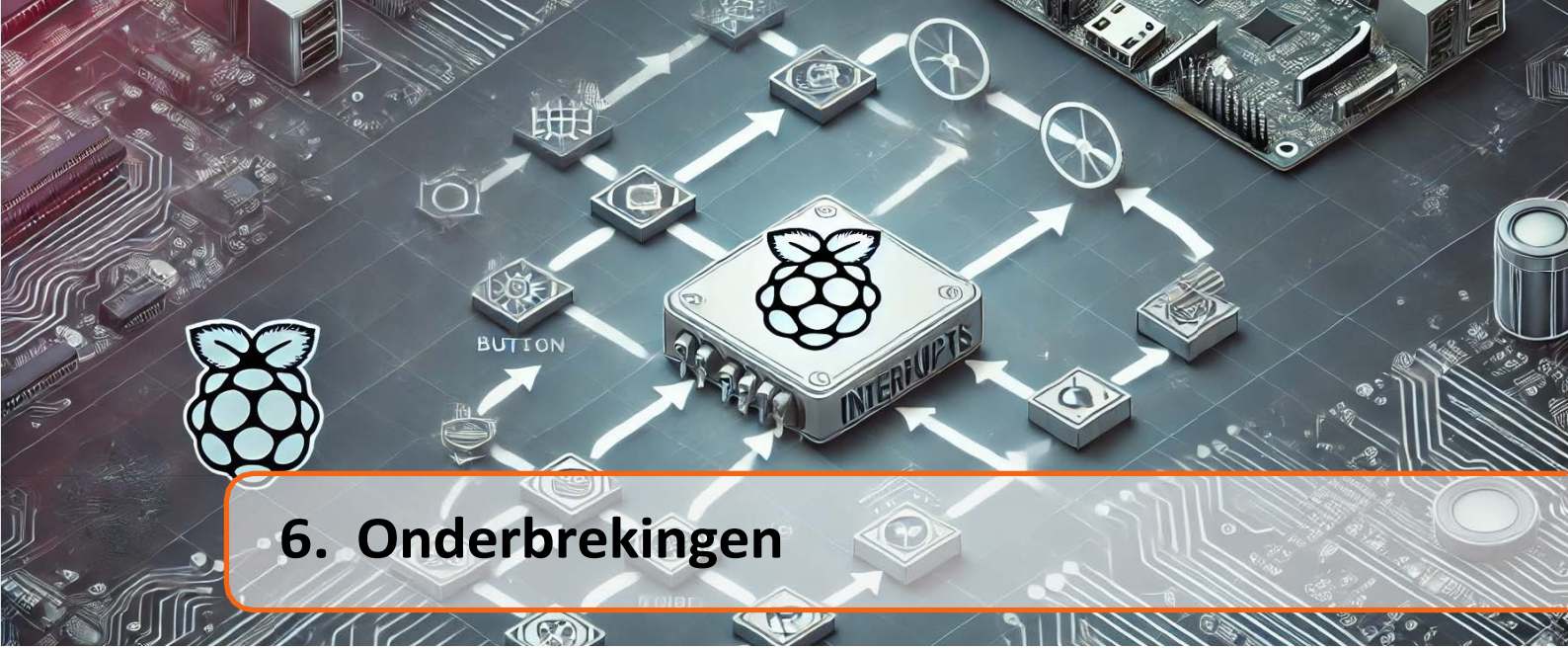
```
13
14          utime. slaap (1)
```

8. De laatste stap is om beide resultaten in één regel weer te geven in de terminal:

```
importeer machine
importeer utime

built_in_temp= machine. ADC (4) external_temp=
machine. ADC (26)


terwijl True:
        spanning1   =  built_in_temp . read_u16 () * 3 .3 /65535
            temp1= 27  - ( spanning1 -0,706) /0,001721

        spanning2   =  extern_temp . read_u16 () * 3 .3 /65535
        temp2=
        Temp1 ="+ str( temp1 )+ " Temp2 ="+ str( temp2 )) utime.
        sleep (1)
```

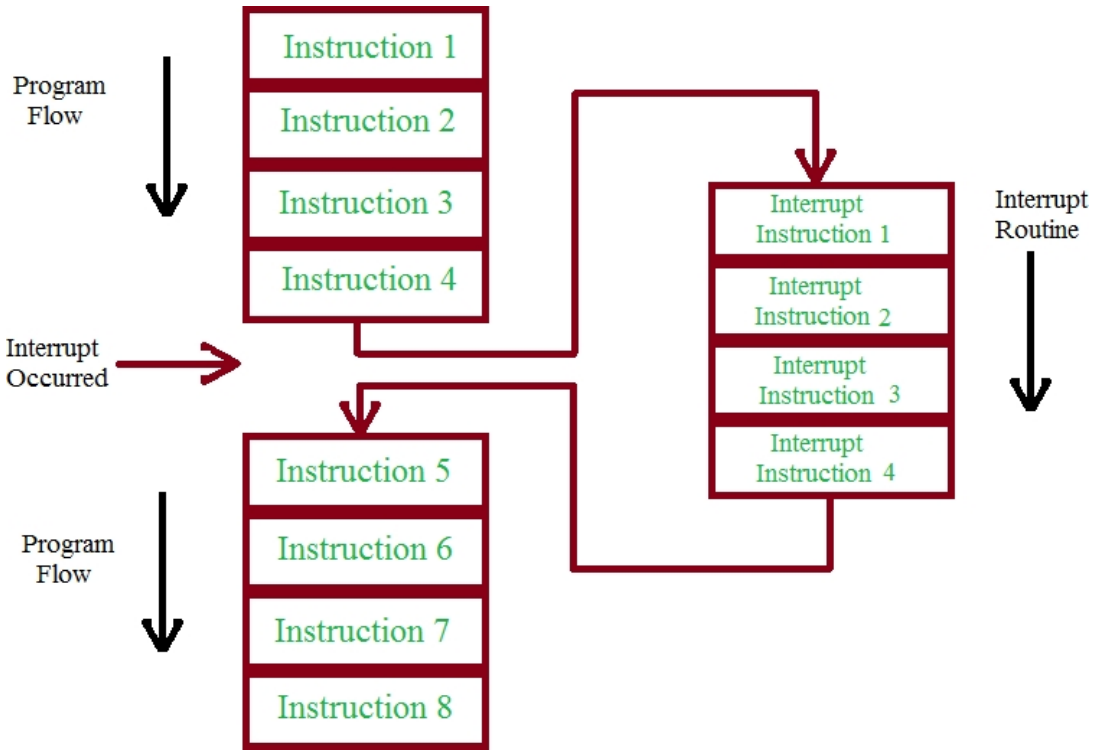
Test het programma en vergelijk de resultaten.

Het gebruik van externe ADC's om nauwkeurigere spanningsmetingen te verkrijgen wordt later in deze tutorial besproken.



6. Onderbrekingen

Interrupts geven ons een mechanisme dat het mogelijk maakt om de uitvoering van een programma te stoppen voor de tijd van een speciale procedure en dan terug te keren naar het moment waar het programma gestopt was en het te hervatten (zie Fig. 6.1). Interrupts kunnen worden getriggerd door externe of interne signalen.



Afbeelding 6.1: Hoe interrupts werken. Bron: https://makergram.com/blog/what-is-interrupts/

Om het idee beter te begrijpen, worden onderbrekingen automatisch uitgevoerd op veel momenten in het leven, bijvoorbeeld als we een boek aan het lezen zijn en de telefoon gaat, stoppen we met het lezen van het boek.

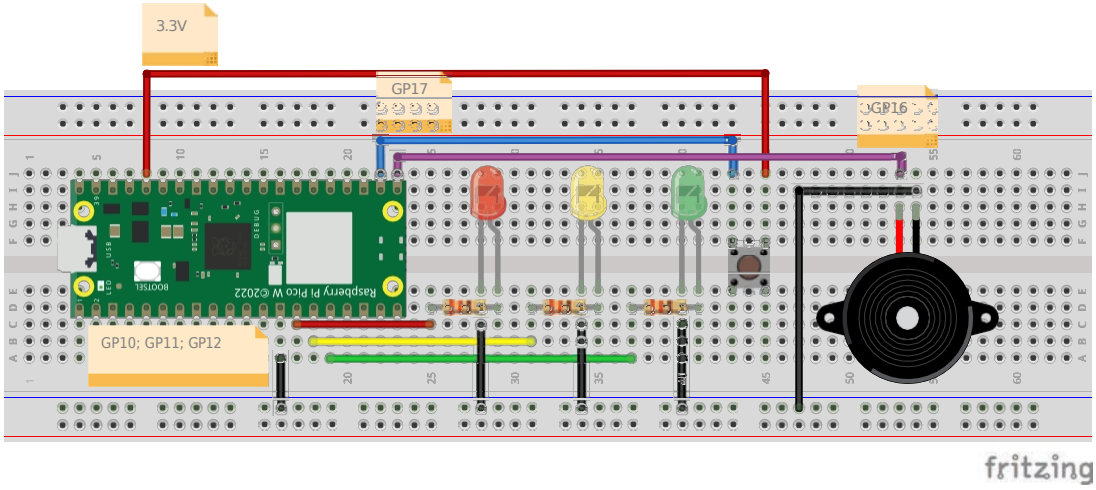
om de telefoon op te nemen en nadat het gesprek is beëindigd, gaan we weer verder met het lezen van het boek. Precies hetzelfde idee doet zich voor bij microcontrollers. Externe interrupts kunnen worden getriggerd door de volgende signalen op de geselecteerde GPIO pin:

- Pin.IRQ_RISING - als het signaal verandert van laag naar hoog.
- Pin.IRQ_FALLING - als het signaal van hoog naar laag gaat.

6.1 Voorbeeld 6: Geluidssignalen bij verkeerslichten

Om het idee van onderbreking beter te begrijpen, zullen we een voetgangersstoplicht maken met een geluidssignaal voor blinden en slechtzienden wanneer er op een knop wordt gedrukt. Hiervoor gebruiken we 3 LED's om het verkeerslicht aan te geven, een zoemer met een toongenerator en een knop die het geluidssignaal inschakelt voor minstens één volledige reeks lichten.

Sluit het systeem eerst aan volgens Fig. 6.2.



Afbeelding 6.2: Het aansluiten van het elektronische circuit uit voorbeeld 6.

Open dan de Thonny editor en volg deze stappen:

1. Voeg eerst de benodigde bibliotheken toe: *machine* en *utime* en configureer de pinnen waarop de LED's zijn aangesloten als uitgangen:

```
importeer machine
importeer utime

led_red= machine . Pin (10 , machine . Pin. OUT )
led_yellow= machine. Pin (11 , machine. Pin. OUT )
led_green= machine. Pin (12 , machine. Pin. OUT )
```

2. Configureer vervolgens de GP16-pin waarop de zoemer is aangesloten als een pin voor het genereren van een PWM-signaal en stel de PWM-signaalfrequentie in op 1000 Hz.

```
importeer machine
importeer utime

led_red= machine . Pin (10 , machine . Pin. OUT )
led_yellow= machine. Pin (11 , machine. Pin. OUT )
```

```
led_green= machine. Pin (12 , machine. Pin. OUT )

zoemer= machine. PWM ( machine. Pin (16)) zoemer.
freq (1000)
```

3. In de hoofdlus van het programma moeten de LED's branden volgens de volgorde:
   - rode LED aan, andere LED's uit
   - rode en gele LED's aan, groene uit
   - groene LED aan en andere LED's uit.

```
importeer machine
importeer utime

led_red= machine . Pin (10 , machine . Pin. OUT )
led_yellow= machine. Pin (11 , machine. Pin. OUT )
led_green= machine. Pin (12 , machine. Pin. OUT )

zoemer= machine. PWM ( machine. Pin (16))
zoemer. freq (1000)

terwijl True:
    led_red . waarde (1)
    led_geel . waarde (0)
    led_groen . waarde (0)

    led_red . waarde (1)
    led_geel . waarde (1)
    led_groen . waarde (0)

    led_red . waarde (0)
    led_geel . waarde (0)
    led_groen . waarde (1)

```

4. Laten we vervolgens een logische variabele *sound_active* maken, die een van de twee mogelijke waarden heeft: *True* of *False*. Als de waarde *True is*, starten we de zoemer en genereren we een geluidssignaal dat aangeeft of de lichten rood of groen zijn. Laten we daarnaast een functie *make_sound(duration)* maken, waarin we een geluidssignaal genereren. Hiervoor gebruiken we de functie *duty_u16()*, die de duty van het PWM-signaal instelt. Deze functie accepteert waarden van 0 tot 65535. De verandering van de duty heeft invloed op het volume van het geluid dat door de zoemer wordt gegenereerd. Geluidssignalen in verschillende landen kunnen verschillen, maar als regel wordt aangenomen dat tijdens een rood of geel licht het geluidssignaal met langere intervallen wordt gegenereerd dan tijdens een groen licht. Om de intervallen tussen de gegenereerde geluidssignalen te regelen, maken we een variabele *duration* als argument voor de functie:

```
importeer machine
importeer utime

led_red= machine . Pin (10 , machine . Pin. OUT )
led_yellow= machine. Pin (11 , machine. Pin. OUT )
led_green= machine. Pin (12 , machine. Pin. OUT )

```

```
zoemer= machine. PWM ( machine. Pin (16)) zoemer.
freq (1000)



def make_sound(duration): if
      sound_active :
            buzzer. duty_u16 (16383) % zoemer aanzetten
            utime. sleep (1)
            buzzer. duty_u16 (0) % buzzer uitzetten utime.
            sleep( duur )
      anders:
            zoemer. duty_u16 (0)

            # Omdat wanneer het geluidssignaal wordt aangezet , #
            de totale vertraging 1+ duur is . Dus hier # voegen we
            1 toe om dezelfde vertraging te behouden .

            utime. sleep( duur +1) while

True:
      led_red . waarde (1)
            ...
```

5. ~~Laten we nu geluid genereren na elk verkeerslicht. Het apparaat moet 5 keer een~~ waarschuwingssignaal genereren. Hiervoor wordt een for-lus gemaakt die 5 keer wordt uitgevoerd en de functie *make_sound* 5 keer aanroept. In het geval van rood of geel moet het interval tussen de geluidssignalen 2 seconden zijn en in het geval van groen 1 seconden:

```
importeer machine
importeer utime

led_red= machine . Pin (10 , machine . Pin. OUT )
led_yellow= machine. Pin (11 , machine. Pin. OUT )
led_green= machine. Pin (12 , machine. Pin. OUT )

zoemer= machine. PWM ( machine. Pin (16)) zoemer.
freq (1000)

geluid_actief= Vals

def make_sound ( duration ): if
      sound_active :
            buzzer. duty_u16 (16383) % zoemer aanzetten
            utime. sleep (1)
            buzzer. duty_u16 (0) % buzzer uitzetten utime.
            sleep( duur )
      anders:
            zoemer. duty_u16 (0)
```

```
            utime. sleep( duur +1)

terwijl True:
      led_red . waarde (1)
      led_yellow . waarde (0)
      led_green . waarde (0)
      voor i in bereik (0,5):
            make_sound (2)

      led_red . waarde (1)
      led_yellow . waarde (1)
      led_green . waarde (0)
      voor i in bereik (0,5):
            make_sound (2)

      led_red . waarde (0)
      led_yellow . waarde (0)
      led_green . waarde (1)
      voor i in bereik (0,5):
            make_sound (1)
```

6. Laten we nu de GP17 pin, waarop de knop is aangesloten, configureren als een ingang. Laten we bovendien de interrupt configureren. Gebruik hiervoor de functie: ***irq(trigger, handler)***. Het eerste argument specificeert voor welk type signaal de interrupt gevoelig is, bijvoorbeeld een dalende of stijgende flank. In het geval van een opgaande flank, zal de interrupt worden getriggerd wanneer de knop wordt ingedrukt, en in het geval van een neergaande flank, wanneer de knop wordt losgelaten. In dit geval gebruiken we een opgaande flank. Het tweede argument is de naam van de functie die we hebben gemaakt en die wordt uitgevoerd wanneer een interrupt optreedt. In de volgende stap maken we een functie met de naam *sound_for_blind*:

```
importeer machine
importeer utime

led_red= machine . Pin (10 , machine . Pin. OUT )
led_yellow= machine. Pin (11 , machine. Pin. OUT )
led_green= machine. Pin (12 , machine. Pin. OUT )

zoemer= machine. PWM ( machine. Pin (16)) zoemer.
freq (1000)

geluid_actief= Vals

def make_sound ( duration ): if
      sound_active :
            buzzer. duty_u16 (16383) % zoemer aanzetten
            utime. sleep (1)
            buzzer. duty_u16 (0) % buzzer uitzetten utime.
            sleep( duur )
      anders:
            buzzer. duty_u16 (0) utime.
            sleep( duur +1)
```

```
knop= machine.Pin(17, machine.Pin.IN, machine.Pin.PULL_DOWN
button.irq(trigger=machine.Pin.IRQ_RISING, handler=geluid_voor_blind

terwijl True:
    led_red . waarde (1)
    ...
```

7. Laten we nu een functie maken die wordt aangeroepen wanneer een interrupt optreedt. In deze functie wijzigen we de waarde van de variabele *sound_active* in True. Aangezien dit een variabele is die buiten de functie is gedefinieerd, moeten we het commando global *variable_name* oproepen, dat aangeeft dat deze variabele buiten de functie is gemaakt:

```
importeer machine
importeer utime

led_red= machine . Pin (10 , machine . Pin. OUT )
led_yellow= machine. Pin (11 , machine. Pin. OUT )
led_green= machine. Pin (12 , machine. Pin. OUT )

zoemer= machine. PWM ( machine. Pin (16))
zoemer. freq (1000)

geluid_actief= Vals

def make_sound ( duration ): if
    sound_active :
        buzzer. duty_u16 (16383) % zoemer aanzetten
        utime. sleep (1)
        buzzer. duty_u16 (0) % buzzer uitzetten utime.
        sleep( duur )
    anders:
        buzzer. duty_u16 (0)
        utime. sleep( duur +1)

def geluid_voor_blind(pin):
    global geluid_actief
    geluid_actief= Waar
    print(" Geluid    actief: "   + str( geluid_actief))

knop= machine. Pin (17 , machine. Pin .IN , machine. Pin .
    →PULL_DOWN )
knop . irq( trigger= machine . Pin. IRQ_RISING ,   handler=
    →geluid_voor_blinden)

terwijl True:
    led_red . waarde (1)
    ...
```

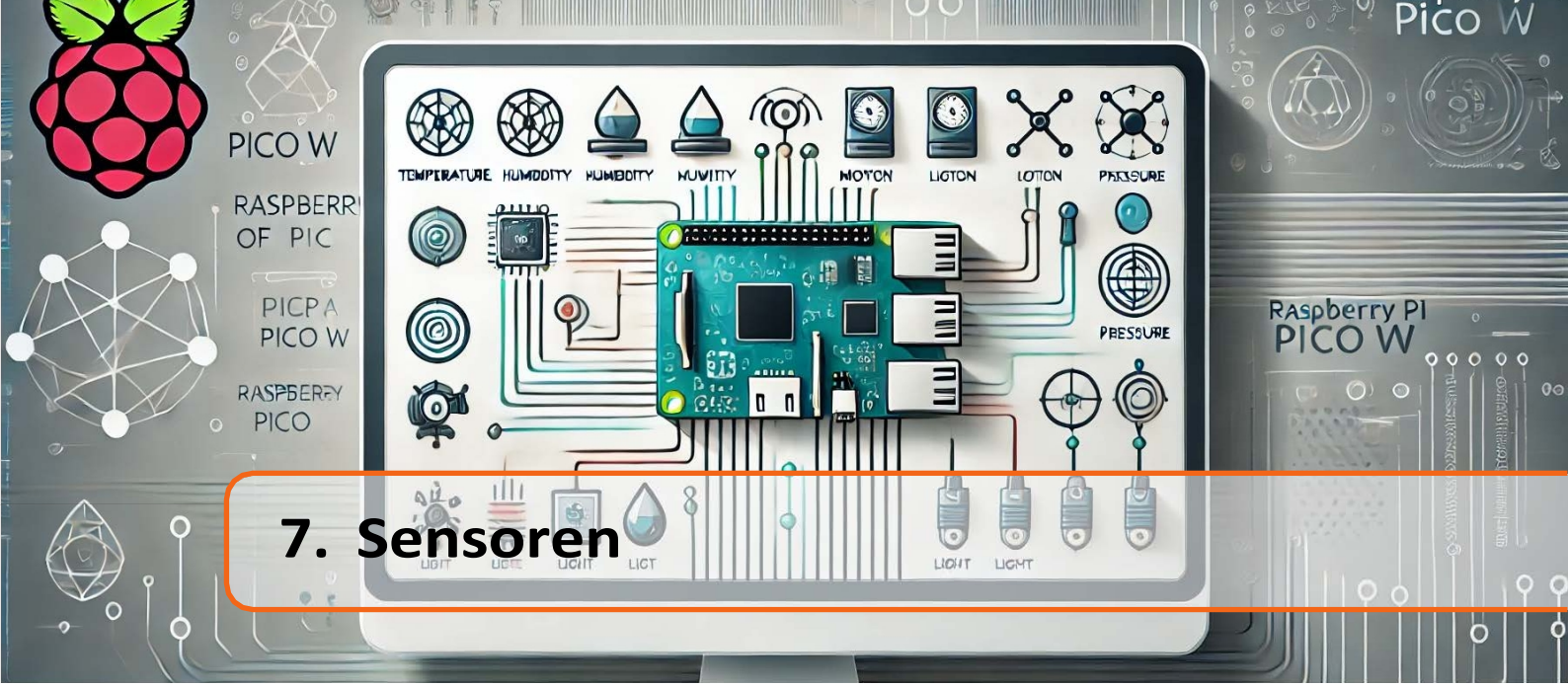
Als de gebruiker nu op de knop drukt, wordt de functie *sound_for_blind* aangeroepen, ongeacht welke regel code in de while-lus het programma zal uitvoeren.

8. De laatste stap is het uitschakelen van het geluidssignaal na minstens één volledige lichtschakelsequentie. Om dit te doen, maken we een variabele *counts* aan die telt hoeveel keer de volledige sequentie is uitgevoerd na de onderbreking. Vervolgens controleren we in het hoofdprogramma of *counts* groter of gelijk is aan 1. Zo ja, dan schakelen we het geluidssignaal uit door de variabele *sound_active* op *False* te zetten. Als het geluidssignaal aan is, dan verhogen we bij elke iteratie van de while-lus de waarde van de variabele *counts* met 1:

```
importeer machine
importeer utime

led_red= machine . Pin (10 , machine . Pin. OUT )
led_yellow= machine. Pin (11 , machine. Pin. OUT )
led_green= machine. Pin (12 , machine. Pin. OUT )

zoemer= machine. PWM ( machine. Pin (16)) zoemer.
freq (1000)

geluid_actief= Vals

def make_sound ( duration ): if
      sound_active :
            buzzer. duty_u16 (16383)
            utime. sleep (1) buzzer.
            duty_u16 (0) utime.
            sleep( duur )
      anders:
            buzzer. duty_u16 (0) utime.
            sleep( duur +1)

def geluid_voor_blind( pin):
      globaal geluid_actief
      geluid_actief= Waar
      print(" Geluid    actief: "   + str( geluid_actief))

knop= machine. Pin (17 , machine. Pin .IN , machine. Pin .
    →PULL_DOWN )
knop . irq( trigger= machine . Pin. IRQ_RISING ,   handler=
    →geluid_voor_blinden)

counts=0 while

True:
#Hoe vaak de volledige reeks is
      als tellingen >=1:
            sound_active= False counts =0
      als geluid_actief : telt
            =+1
```

```
41          led_red . waarde (1)
42          led_yellow . waarde (0)
43          led_green . waarde (0)
44          voor i in bereik (0,5):
45              make_sound (2)
46
47          led_red . waarde (1)
48          led_yellow . waarde (1)
49          led_green . waarde (0)
50          voor i in bereik (0,5):
51              make_sound (2)
52
53          led_red . waarde (0)
54          led_yellow . waarde (0)
55          led_green . waarde (1)
56          voor i in bereik (0,5):
57              make_sound (1)
```

Het programma is nu klaar, je kunt het testen.



7. Sensoren

Sensoren worden gebruikt om verschillende fysische grootheden te meten, zoals temperatuur, druk, intensiteit van UV-straling, versnelling, geluidsdetectie, gasconcentratie, enz. Sensoren kunnen analoge waarden teruggeven (bijv. temperatuursensor besproken in hoofdstuk 5) of digitale waarden. Digitale waarden kunnen 0-1 zijn (bijv. bewegingssensor, geluidsdetectiesensor) of het kunnen waarden uit een bepaald bereik zijn, waarna seriële communicatie-interfaces (bussen) zoals *I²C*, *SPI*, *UART* of 1− *draad* worden gebruikt om gegevens over te dragen. Digitale transmissie heeft het voordeel ten opzichte van analoge transmissie dat het veel minder gevoelig is voor externe interferentie en ruis.

De seriële communicatie-interface bestaat uit een groep lijnen die worden gebruikt om gegevens te verzenden tussen aangesloten apparaten. Seriële interfaces kunnen in twee groepen worden verdeeld: synchroon en asynchroon. De eerste groep gebruikt een extra seriële kloklijn, die alle apparaten die zijn aangesloten op de communicatie-interface synchroniseert. De populairste synchrone bussen zijn *SPI* (Serial Peripheral Interface) en *I²C* (Inter-Integrated Circuit), ook wel aangeduid als *I2C*, *IIC* of *TWI* (Two Wire Interface). De populairste asynchrone seriële communicatie-interfaces zijn *UART* (Universal asynchronous receiver-transmitter) en 1− *wire*.

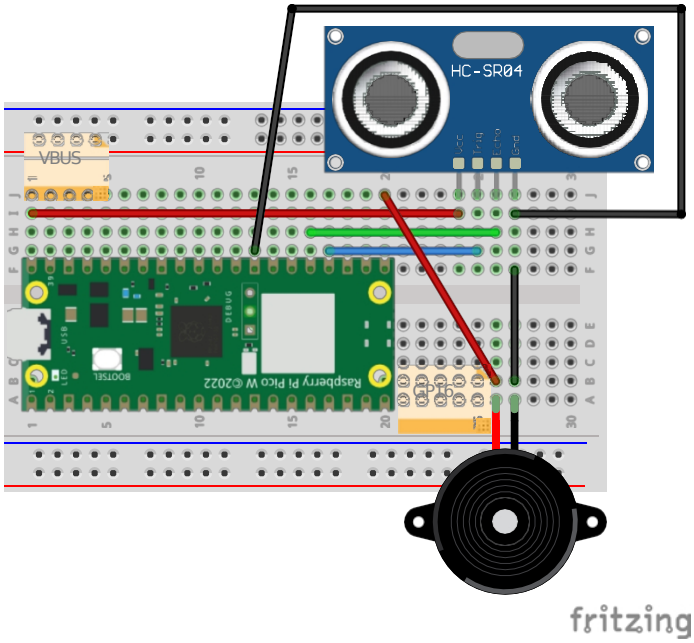
In dit hoofdstuk bespreken we één sensor per type digitale communicatie. Andere sensoren worden op de meeste manieren op dezelfde manier gebruikt als de sensoren die hieronder worden besproken.

7.1 Voorbeeld 7: parkeerplaats sensor

Parkeersensoren maken geluid wanneer de afstand tussen de auto en het obstakel klein wordt, bijvoorbeeld minder dan 1 meter bij achteruit inparkeren. De frequentie van het uitgezonden geluid wordt hoger naarmate de afstand tot het obstakel kleiner is. Om een parkeersensor te maken, gebruiken we de HC-SR04 ultrasone afstandssensor en een zoemer. Sluit het systeem eerst aan volgens Fig. 7.1.

Ga nu naar Thonny editor en begin met het maken van een programma dat eerst de afstand tot een obstakel leest met behulp van een ultrasone afstandssensor en vervolgens een waarschuwingsgeluid genereert als het obstakel dichterbij is dan 1 meter. Volg hiervoor deze stappen:

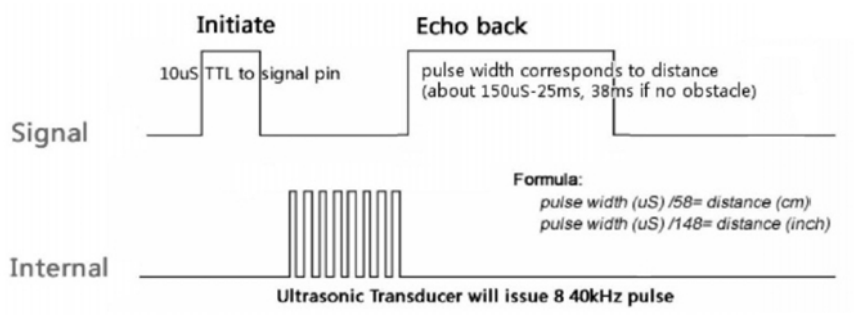
1. Laten we eerst de benodigde bibliotheken toevoegen, namelijk *machine* en *utime*, en de pinnen configureren: GP16 (zoemer) als PWM, GP18 (trigger) als uitgang en GP19 (echo) als ingang. Daarnaast stellen we de frequentie voor het genereren van het PWM-signaal in op 1000 Hz.



Figuur 7.1: Het aansluiten van het elektronische circuit uit voorbeeld 7.

```
importeer machine
importeer utime

trigger= machine . Pin (18 , machine . Pin . OUT ) echo=
machine. Pin (19 , machine. pin . IN)

zoemer= machine. PWM ( machine. Pin (16))
zoemer. freq (1000)
```

2. Laten we vervolgens in de hoofdlus een codefragment toevoegen om de afstand tot de ultrasone afstandssensor te meten. Volgens de documentatie voor de HC-SR04 sensor (zie Fig. 7.2), stel je eerst het lage signaal op de trigger in voor een korte tijd, bijvoorbeeld *2µs*. Stel dan het hoge signaal in voor *10* µs. Gebruik de functie: *utime.sleep_us()* om vertragingen in microseconden te genereren. Stel in de volgende stap het lage signaal in op de trigger.



Afbeelding 7.2: Principe van afstandsmeting bij de HC-SR04 ultrasone afstandssensor. Bron: https://www.electronicoscaldas.com/datasheet/HC-SR04.pdf.

```
importeer machine
importeer utime

trigger= machine . Pin (18 , machine . Pin . OUT ) echo=
machine. Pin (19 , machine. pin . IN)

zoemer= machine. PWM ( machine. Pin (16))
zoemer. freq (1000)

terwijl True:
      trigger. waarde (0)
      utime. sleep_us (2)
      trigger. waarde (1)
      utime. sleep_us (10)
      trigger. waarde (0)
```

3. Nu moeten we meten hoe lang het hoge signaal op de echo pin duurde, omdat de duur van het signaal op de echo pin gerelateerd is aan de afstand. Om dit te doen zullen we de functie *utime.ticks_us()* gebruiken, die meet hoeveel tijd er is verstreken in μs sinds het programma werd gestart. Eerst maken we een while-lus die wordt uitgevoerd zolang het signaal laag is. Daarbinnen plaatsen we de functie *tick_us()*. Op deze manier krijgen we informatie over wanneer het signaal voor het laatst laag was.

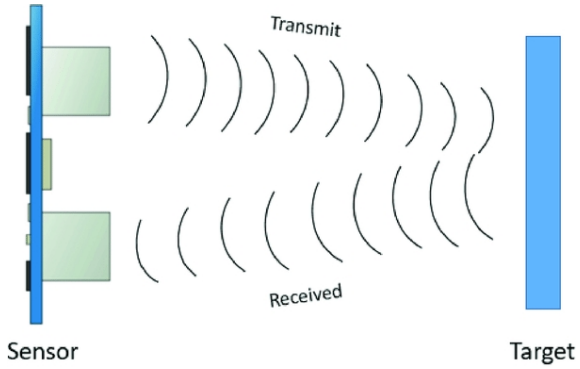
```
importeer machine
importeer utime

trigger= machine . Pin (18 , machine . Pin . OUT ) echo=
machine. Pin (19 , machine. pin . IN)

zoemer= machine. PWM ( machine. Pin (16))
zoemer. freq (1000)

terwijl True:
      trigger. waarde (0)
      utime. sleep_us (2)
      trigger. waarde (1)
      utime. sleep_us (10)
      trigger. waarde (0)

      terwijl   echo. waarde () ==0:
            signal_off= utime. ticks_us ()
```

4. Op dezelfde manier zullen we meten wanneer het signaal voor het laatst hoog was op de echo-pen:

```
importeer machine
importeer utime

trigger= machine . Pin (18 , machine . Pin . OUT )
echo= machine. Pin (19 , machine. pin . IN)

zoemer= machine. PWM ( machine. Pin (16))
```

```
zoem. freq (1000)

terwijl True:
    trigger. waarde (0)
    utime. sleep_us (2)
    trigger. waarde (1)
    utime. sleep_us (10)
    trigger. waarde (0)

    terwijl   echo. waarde () ==0:
        signal_off= utime. ticks_us ()

    terwijl   echo. waarde () ==1:
        signal_on  =  utime. ticks_us ()
```

5. Het verschil tussen de tijd van het laatste hoog en laag signaal is de duur van de hoge puls op de echo pin. Hoe vertalen we de pulsduur in afstand? Kijk naar Fig. 7.3, waarin het idee wordt getoond van het meten van afstand met een ultrasone afstandssensor.



Afbeelding 7.3: Het principe van afstandsmeting met een ultrasone sensor.... Bron: https://www.researchgate.net/figure/A-block-diagram-of-Ultrasonic-se nsor-werking-principes_fig5_344385811.

Aan het begin wordt een geluidsgolf uitgezonden die door het object weerkaatst en terugkeert naar de sensor. Daarom legt de golf in de gemeten tijd t tweemaal de afstand tussen de sensor en het object af en beweegt met een snelheid van ongeveer 340 m/s (de geluidssnelheid in lucht). Daarom kunnen we de vergelijking voor de snelheid schrijven:

$$v=\frac{2d}{t} \tag{7.1}$$

$$d= v \cdot \frac{t}{2} = 0.034 \frac{cm}{\mu s} \frac{t}{2} \approx \frac{t}{58} \tag{7.2}$$

Daarom moet de verkregen pulsduur worden gedeeld door 58 om de afstand in centimeters te krijgen:

```
importeer machine
importeer utime

trigger= machine . Pin (18 , machine . Pin . OUT ) echo=
machine. Pin (19 , machine. pin . IN)

zoemer= machine. PWM ( machine. Pin (16))
zoemer. freq (1000)

terwijl True:
    trigger. waarde (0)
    utime. sleep_us (2)
    trigger. waarde (1)
    utime. sleep_us (10)
    trigger. waarde (0)

    terwijl   echo. waarde () ==0:
        signal_off= utime. ticks_us ()

    terwijl   echo. waarde () ==1:
        signal_on  =  utime. ticks_us ()

    diff= signaal_aan - signaal_uit
    afstand= diff /58,0
    print(" Afstand="+ str( afstand))
```

6. De laatste stap is het genereren van een waarschuwingsgeluid op de zoemer. controleren we of de afstand minder dan 100 cm is. Als dat zo is, genereren we een PWM-signaal met de geselecteerde duty cycle op de zoemer, waardoor een geluidsgolf wordt uitgezonden. De ingevoerde waarde vertaalt zich in het volume van het uitgezonden geluid. Vervolgens stellen we een vertraging in die evenredig is met de afstand. In dit geval werd de afstand gedeeld door 50 zodat het waarschuwingsgeluid niet te lang was. De waarde van 50 is empirisch gekozen, het is niet erg logisch. Vervolgens zetten we de zoemer uit door de vulling op 0 te zetten en wachten we ook een tijd die evenredig is met de afstand.

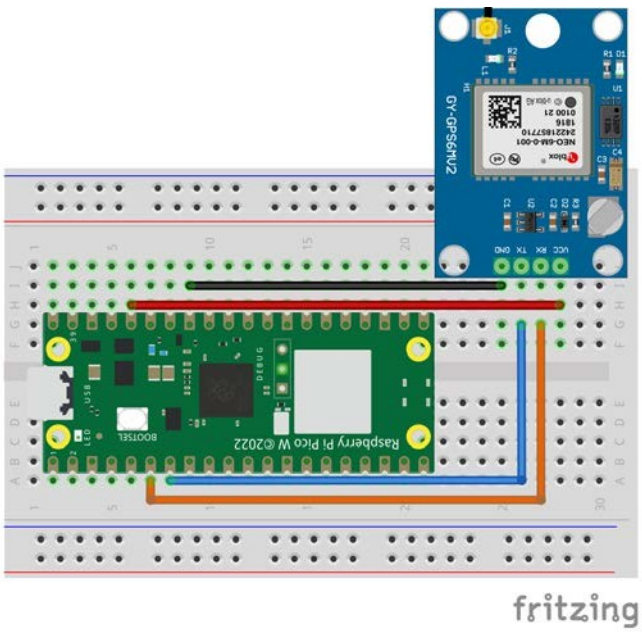
```
importeer machine
importeer utime

trigger= machine . Pin (18 , machine . Pin . OUT )
echo= machine. Pin (19 , machine. pin . IN)

zoemer= machine. PWM ( machine. Pin (16)) zoemer.
freq (1000)

terwijl True:
    trigger. waarde (0)
    utime. sleep_us (2)
    trigger. waarde (1)
    utime. sleep_us (10)
    trigger. waarde (0)
```

```
16
17      terwijl   echo. waarde () ==0:
18            signal_off= utime. ticks_us ()
19
20      terwijl   echo. waarde () ==1:
21            signal_on  =   utime. ticks_us ()
22
23      diff= signaal_aan - signaal_uit
24      afstand= diff /58,0
25      print(" Afstand="+ str( afstand))
26
27      als afstand <100:
28            buzzer. duty_u16 (16383)
29            utime. sleep( distance /50)
30            buzzer. duty_u16 (0) utime.
31            sleep( distance /50)
```

Nu is het programma klaar en kun je het testen.

7.2 Voorbeeld 8: GPS-module

In dit voorbeeld zullen we ons richten op de asynchrone UART communicatiebus, die gebruikt zal worden om gegevens te ontvangen van de Waveshare Neo-6m/7m GPS-module. De Raspberry Pi Pico heeft 2 UART-poorten: UART0 en UART1 (zie Fig: 1.1). We sluiten de GPS-module aan op UART1 (GP4 en GP5 pinnen). Vergeet niet dat de lijnen naar Rx (Ontvanger) en Tx (Zender) kruislings moeten worden aangesloten, d.w.z. de Rx-lijn van de GPS-module naar de Tx van de Raspberry Pi Pico (GP4) en de Tx-lijn van de GPS naar de Rx van de Raspberry Pi Pico (GP5). We sluiten de voeding van de GPS-module aan op 3,3V. Sluit het systeem dus aan zoals getoond in Fig. 7.4.



Afbeelding 7.4: Het aansluiten van het elektronische circuit uit voorbeeld 8.

Open nu de Thonny editor en laten we proberen code te schrijven die dataframes ontvangt van de GPS module:

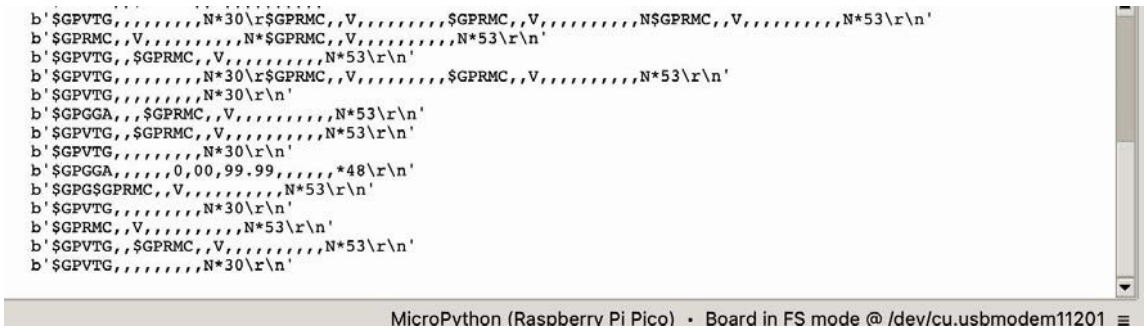
1. Eerst moet je de *machine* en *utime* bibliotheken toevoegen en de UART configureren. Gebruik hiervoor de functie *machine.UART()*, die het UART poortnummer als eerste argument neemt, d.w.z. de waarde 0 of 1. We hebben de GPS-module aangesloten op UART1, dus voeren we 1 in. Vervolgens moet je de baudsnelheid opgeven en de pinnen waarop de GPS module is aangesloten. Als je Fig. 1.1 bekijkt, is UART1 beschikbaar op twee sets pinnen: GP4 en GP5 of GP8 en GP9. Je moet dus aangeven welke pinnen de GPS-module wordt aangesloten:

```
importeer machine
importeer utime

uart= machine . UART (1 , baudrate =9600 , tx= machine . Pin (4),
    →rx= machine. Pin (5))
```

2. Dan controleren we in de hoofdlus of er data beschikbaar is in de UART buffer met de *any()* functie en als dat zo is, lezen we de lijn met de *readline()* functie. We tonen de gelezen regel in de terminal:

```
importeer machine
importeer utime

uart= machine . UART (1 , baudrate =9600 , tx= machine . Pin (4),
    →rx= machine. Pin (5))

terwijl True:
    Als uart. any ():
        regel= uart. readline ()
        print( regel)
    utime . sleep (1)
```

Als je het programma uitvoert, krijg je aan het begin het resultaat zoals in Afb. 7.5.



Figuur 7.5: Voorbeeldoutput van GPS-module zonder fix naar de satellieten te krijgen.

Dit resultaat betekent dat de GPS-module nog geen fix naar de satellieten heeft en lege karakters tussen de komma's verstuurt. Wanneer het een fix krijgt, verschijnen er gegevens tussen de komma's die we kunnen interpreteren (zoals in Fig. 7.6). Hoe gegevens van GPS interpreteren? De gegevens van de GPS worden opgeslagen volgens het NMEA-protocol (National Marine Electronics Association), waarin elke

```
$GPGLL,2232.73995,N,11404.60273,E,030427.00,A,A*6B
$GPRMC,030428.00,A,2232.73995,N,11404.60275,E,0.037,,070314,,,A*7E
$GPVTG,,T,,M,0.037,N,0.069,K,A*28
$GPGGA,030428.00,2232.73995,N,11404.60275,E,1,07,1.17,122.5,M,-2.3,M,,*4F
$GPGSA,A,3,29,21,18,05,14,22,26,,,,,,2.12,1.17,1.77*00
$GPGSV,3,1,10,05,18,096,31,12,07,154,15,14,12,248,29,15,44,025,*7B
$GPGSV,3,2,10,18,38,325,43,21,61,296,41,22,09,304,31,24,70,114,*7D
$GPGSV,3,3,10,26,10,045,16,29,16,208,35*7B
$GPGLL,2232.73995,N,11404.60275,E,030428.00,A,A*62
$GPRMC,030429.00,A,2232.73994,N,11404.60277,E,0.017,,070314,,,A*7E
$GPVTG,,T,,M,0.017,N,0.031,K,A*27
$GPGGA,030429.00,2232.73994,N,11404.60277,E,1,07,1.17,122.7,M,-2.3,M,,*4F
$GPGSA,A,3,29,21,18,05,14,22,26,,,,,,2.12,1.17,1.77*00
$GPGSV,3,1,10,05,18,096,31,12,07,154,14,14,12,248,29,15,44,025,*7A
$GPGSV,3,2,10,18,38,325,43,21,61,296,41,22,09,304,31,24,70,114,21*7E
$GPGSV,3,3,10,26,10,045,14,29,16,208,35*79
$GPGLL,2232.73994,N,11404.60277,E,030429.00,A,A*60
```

Figuur 7.6: Voorbeelduitvoer van GPS, die een fix naar de satellieten heeft gekregen. Bron: https://www.waveshare.com/wiki/File:UART-GPS-NEO-6M-User-Manual-2.png.

sequentie begint met een identificatie, bv. GPGGA - Global Positioning System Fix Data. Om de interessante gegevens te lezen, moet je dus de juiste sequentie-ID vinden waarin de informatie die je zoekt zich bevindt. In dit geval willen we de positie lezen, dus zijn we alleen geïnteresseerd in het GPGGA-frame. De rest van de frames worden hier besproken: https://aprs.gids.nl/nmea/ . Het GPPGA frame ziet eruit als in Fig. 7.7.

Name	Example Data	Description
Sentence Identifier	$GPGGA	Global Positioning System Fix Data
Time	170834	17:08:34 Z
Latitude	4124.8963, N	41d 24.8963' N or 41d 24' 54" N
Longitude	08151.6838, W	81d 51.6838' W or 81d 51' 41" W
Fix Quality: - 0 = Invalid - 1 = GPS fix - 2 = DGPS fix	1	Data is from a GPS fix
Number of Satellites	05	5 Satellites are in view
Horizontal Dilution of Precision (HDOP)	1.5	Relative accuracy of horizontal position
Altitude	280.2, M	280.2 meters above mean sea level
Height of geoid above WGS84 ellipsoid	-34.0, M	-34.0 meters
Time since last DGPS update	blank	No last update
DGPS reference station id	blank	No station id
Checksum	*75	Used by program to check for transmission errors

Figuur 7.7: GPGGA-sequentie. Bron: https://aprs.gids.nl/nmea/

3. Om een GPGGA frame uit te pakken, moet eerst de gelezen tekst in de vorm van bytes omgezet worden naar een string met utf-8 codering. Hiervoor wordt de functie *decode()* gebruikt. Vervolgens geven we alleen die regels weer die beginnen met $GPGGA. Hiervoor gebruiken we de functie *startswith()*, die *True* teruggeeft als de string begint met het geselecteerde woord tussen haakjes:

```
importeer machine
importeer utime

uart= machine . UART (1 , baudrate =9600 , tx= machine . Pin (4),
    →rx= machine. Pin (5))

terwijl True:
      Als uart. any ():
            regel= uart. readline ()
            regel= regel. decode('utf -8 ') als
            regel. startswith(' $GPGGA '):
                  print( regel)
       utime. sleep (1)
```

4. Vervolgens moeten we de string opsplitsen in fragmenten. Elk stukje informatie in de string wordt gescheiden door een komma, dus gebruiken we de functie *split()*, die de string in fragmenten splitst volgens het scheidingsteken tussen haakjes.

```
importeer machine
importeer utime

uart= machine . UART (1 , baudrate =9600 , tx= machine . Pin (4),
    →rx= machine. Pin (5))

terwijl True:
      Als uart. any ():
            regel= uart. readline ()
            regel= regel. decode('utf -8 ') als
            regel. startswith(' $GPGGA '):
                  parts= line. split(',')
                  latitude= parts [2]
                  lengte  =  delen [4]
                  print( latitude+","+ longitude)
      utime . sleep (1)
```

Nu is het programma klaar en kan het getest worden. Om te controleren of we de juiste locatie hebben gelezen, kunnen we google maps gebruiken. Dan moeten we in het zoekveld de lengte- en breedtegraad invoeren in de vorm van bijv: 52 13.30093, 21 00.41831.

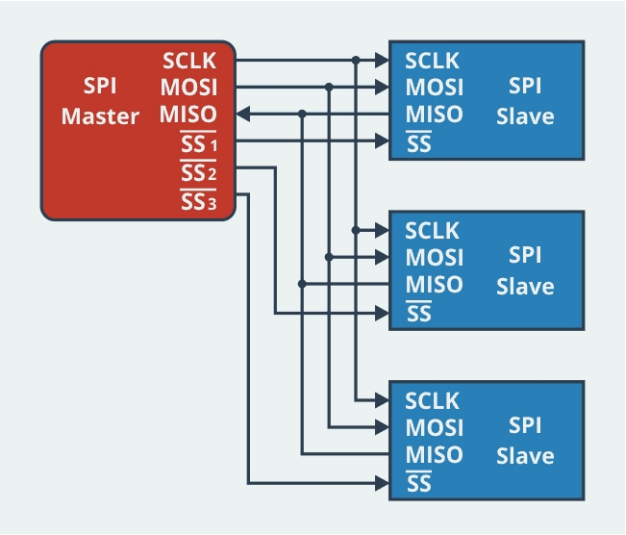
7.3 Voorbeeld 9: Weerstation

De seriële perifere interface bestaat uit vier lijnen:

- SPI SCK - seriële klok, waarmee apparaten gesynchroniseerd kunnen worden,
- SPI TX (vroeger MOSI genoemd - *Master Output Slave Input*) - lijn, die verzendt bits van masterapparaat naar alle slave-apparaten,
- SPI RX (vroeger MISO genoemd - *Master Input Slave Output*) - lijn waarmee bits van slave-apparaten naar master-apparaten worden verzonden,
- SPI CS (Chip select) - lijn die de communicatie met het gekozen slave-apparaat activeert. Het slave-apparaat begint te luisteren en te reageren wanneer een lage waarde wordt ingesteld op
  zijn CS-lijn. Deze lijn wordt soms *Slave select genoemd* en dan gemarkeerd als SS.

Om aan te geven dat de actieve status laag is in de regelnaam wordt vaak een regel boven de tekst geplaatst, zoals *SS* of *CS*.

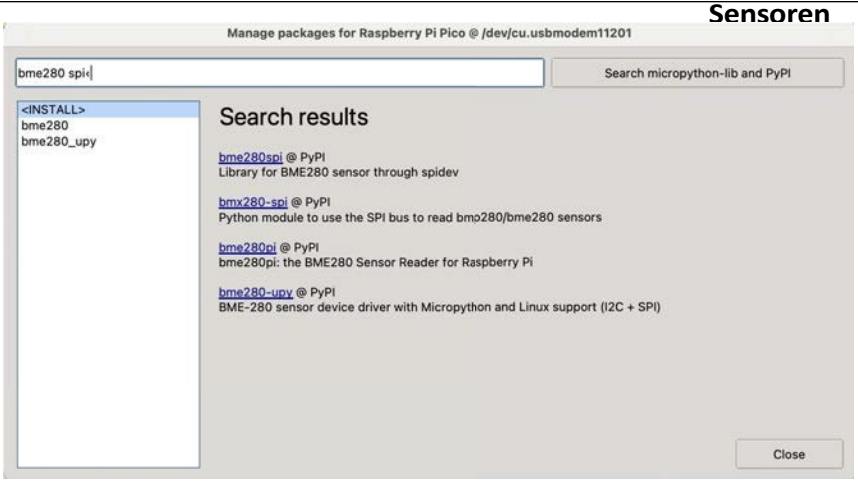
De schematische verbinding tussen apparaten (blokschema) wordt getoond in Fig. 7.8. De master is er maar één en dit apparaat ontvangt op verzoek informatie van de slave-apparaten. De master genereert het kloksignaal dat alle apparaten synchroniseert. Het is mogelijk om veel slave-apparaten (bijvoorbeeld sensoren) aan te sluiten op een master (bijvoorbeeld een Raspberry Pi Pico-bord), maar alle slave-apparaten moeten aparte SS/CS-lijnen hebben. Het is de moeite waard om te vermelden dat SPI een full-duplex bus is. Dit betekent dat gegevens tegelijkertijd verzonden en ontvangen kunnen worden door het apparaat.



Afbeelding 7.8: De schematische verbinding tussen apparaten met SPI-bus. Bron: https://forbot.pl/blog/kurs-stm32-f4-10-obsluga-spi-wyswietlacz-oled-id134 75.

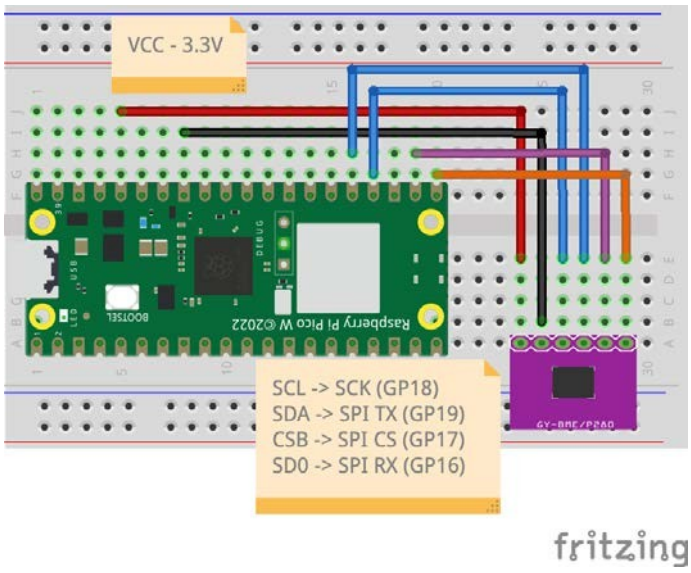
De speciale bibliotheek om de SPI-bus te gebruiken heet *SPI*, maar de meeste sensoren hebben kant-en-klare bibliotheken voor MicroPython, die kunnen worden gevonden in Thonny editor. Dit voorbeeld laat zien hoe je kant-en-klare bibliotheken kunt gebruiken voor sensoren die gegevens verzenden via de SPI-interface.

Als voorbeeld zullen we een weerstation maken gebaseerd op de BME280 sensorwaarmee je temperatuur, druk en vochtigheid kunt meten. Eerst moet je de bibliotheek installeren. In Thonny kunnen we zoeken naar kant-en-klare bibliotheken door *Tools→ Manage Packages* te selecteren in het menu. Vervolgens moet je de naam van de sensor invoeren waarvoor je een bibliotheek zoekt. Aangezien de BME280 zowel via SPI als via I2C data verstuurt, kunt u bij het zoeken naar een bibliotheek ook de busnaam toevoegen, bijvoorbeeld zoals weergegeven in Fig. 7.9.
Selecteer vervolgens een van de kant-en-klare bibliotheken. In dit voorbeeld gebruiken we de ***bme280-upy*** bibliotheek. Klik erop en druk vervolgens op de knop *Installeren*.
Vervolgens moet je de sensor aansluiten op de Raspberry Pi Pico, die 2 onafhankelijke SPI-bussen heeft: SPI0 en SPI1 (zie Fig. 1.1). We zullen SPI0 gebruiken, beschikbaar op pinnen GP16-GP19. Sluit de sensor aan volgens Fig. 7.11.
Bij het installeren van de *bme280-upy* bibliotheek (zie Fig. 7.10), wordt basisinformatie over de bibliotheek gegeven samen met links naar de repository en de PyPI pagina.



Afbeelding 7.9: Een bibliotheek zoeken voor de BME280-sensor.

Afbeelding 7.10: Informatie over de *bme280-upy* bibliotheek.



Afbeelding 7.11: Het aansluiten van het elektronische circuit uit voorbeeld 9.

Open de PyPI-pagina en merk op dat de maker van de bibliotheek meestal een voorbeeld geeft van het gebruik van de bibliotheek, samen met andere noodzakelijke informatie.

Laten we een programma maken dat temperatuur, druk en vochtigheid uitleest van een sensor, gebaseerd op het voorbeeld van de PyPI-pagina van de bme280-upy bibliotheek. Volg hiervoor deze stappen:

a. Voeg eerst de machine en bme280 bibliotheken toe:

```
machine importeren
bme280
```

b. Vervolgens moet je de sensor configureren door aan te geven welke bus je gaat gebruiken voor de communicatie en als het SPI is, moet je ook de gebruikte CS-pin opgeven:

```
machine importeren
bme280

bme= bme280 . BME280 ( spiBus =0 , spiCS =17)
```

c. Vervolgens moet je de gegevens van de sensor uitlezen, wat de temperatuur in graden Celsius oplevert. In het geval van luchtvochtigheid moet je het resultaat met 100 vermenigvuldigen om een resultaat in procenten te krijgen en de druk moet je delen door 100 om een resultaat in hPa te krijgen:

```
machine importeren
bme280

bme= bme280 . BME280 ( spiBus =0 , spiCS =17) while

True:
    temperatuur , vochtigheid , druk= bme. readForced (
        →filter= bme280 . FILTER_2 , =
        →bme280 . OVSMPL_4 ,   humidityOversampling= bme280
        →. OVSMPL_4 , pressureOversampling= bme280 .
        →OVSMPL_4 )
    vochtigheid= vochtigheid *100 druk
    =   druk /100
    print("T="+ str( temperatuur )+", vochtigheid="+ str(
        →vochtigheid )+", druk ="+ str( druk))
```

Voer het programma uit en test hoe het werkt.

7.4 Voorbeeld 10: Luchtkwaliteit binnenshuis meting

De Inter-Integrated Circuit bus bestaat uit twee lijnen:

- SDA (Serial Data Line) is een datalijn die wordt gebruikt om gegevens te verzenden tussen master- en slave-apparaten.
- SCL (seriële kloklijn).

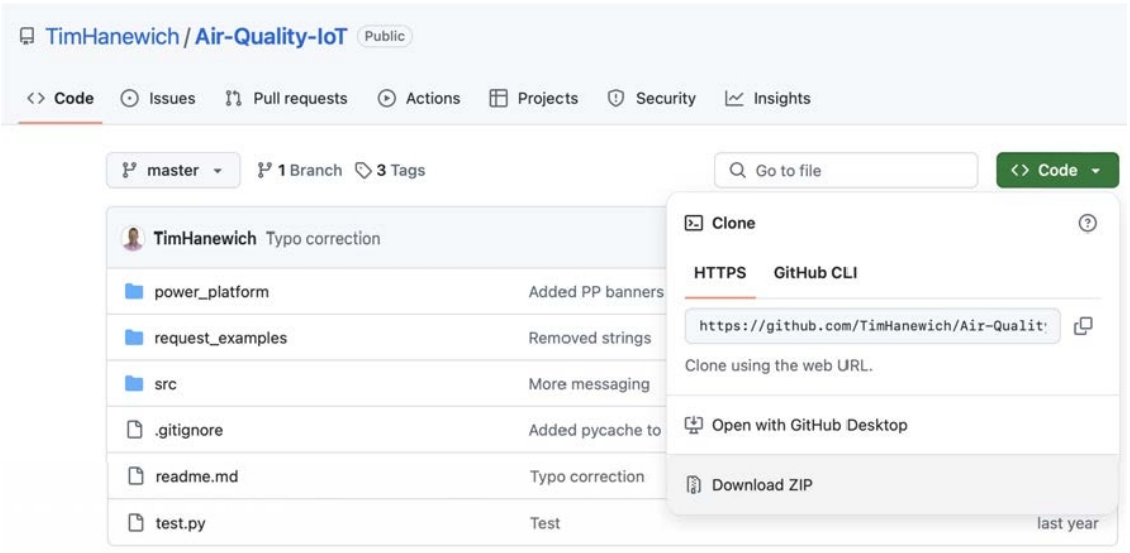
Beide lijnen zijn verbonden met VCC door pull-up weerstanden.

In dit voorbeeld gaan we ons niet verdiepen in het I2C overdrachtsprotocol, maar gebruiken we een kant-en-klare sensorbibliotheek. Merk in Fig. 1.1 op dat we twee I2C-bussen hebben (gemarkeerd als *I2C0* en *I2C1*) die op verschillende pinnen worden uitgestuurd.
Als voorbeeld zullen we een programma maken om luchtkwaliteitsparameters in een kamer uit te lezen, zoals: Luchtkwaliteitsindex, CO2-concentratie (eC02) en totale concentratie vluchtige organische stoffen (TVOC). Hiervoor gebruiken we de DFROBOT ENS160 Luchtkwaliteitssensor. De sensor moet worden aangesloten op de Raspberry Pi volgens tabel nr. 7.1.

ENS160 sensor	Raspberry Pi Pico W
3V3	3V3
GND	GND
SCL	GP15
SDA	GP14

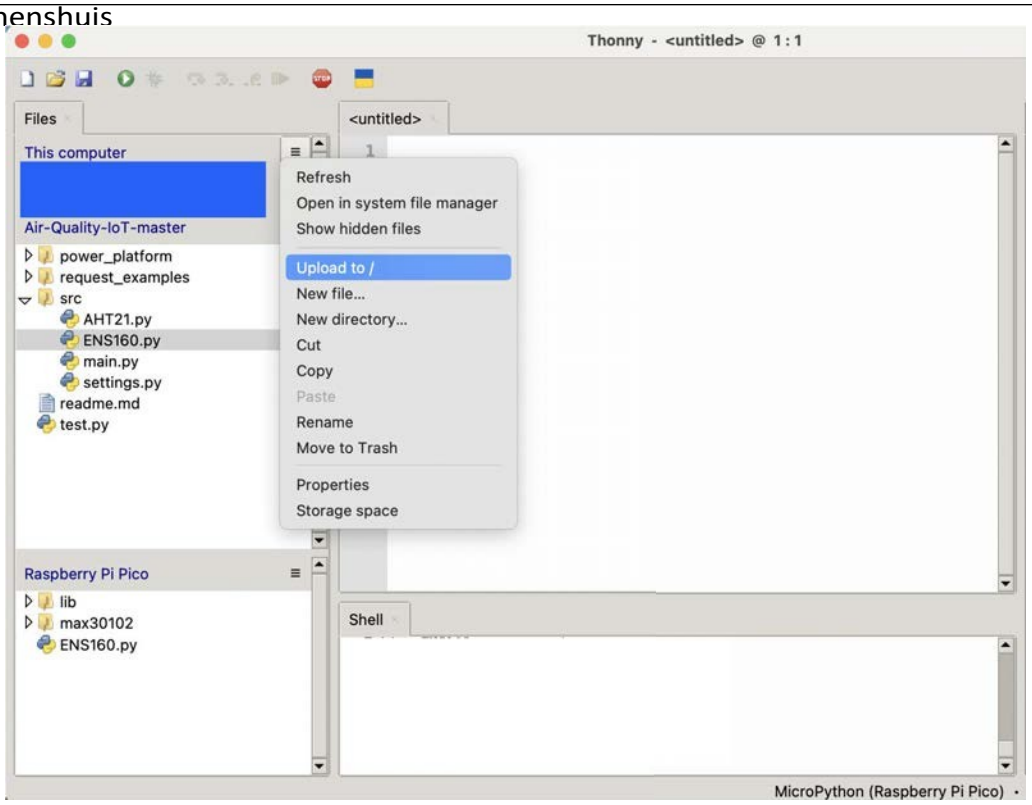
Tabel 7.1: Aansluiting van DFROBOT ENS160 sensor op Raspberry Pi Pico W.

Deze keer vinden we geen kant-en-klare bibliotheek in de Thonny editor package manager. Wat te doen in zo'n geval? Je kunt zoeken naar een kant-en-klare bibliotheek op het internet en deze downloaden, bijvoorbeeld van github. In dit voorbeeld gebruiken we de repository https://github.com/TimHanewich/Air-Quality-IoT/tree/master . Om de kant-en-klare bibliotheek te gebruiken die op github wordt gedeeld, moet je de broncode downloaden door op de knop *Code* te klikken en *ZIP downloaden* te selecteren, zoals in Afb. 7.12 wordt getoond.



Afbeelding 7.12: De bibliotheek downloaden van de github repository.

Vervolgens moet je het zip-bestand uitpakken in de map. Ga naar Thonny en maak verbinding met het Raspberry Pi Pico board en selecteer vervolgens *VIEW→ Files* in het bovenste paneel. Dan krijgen we aan de linkerkant een voorbeeld van de mappen op de computer en daaronder de inhoud van het Raspberry Pi Pico bord. Kopieer de ENS160 bibliotheek (src/ENS160.py) naar de Raspberry Pi Pico zoals getoond in Fig. 7.13.
Meestal staat er in de gedownloade repository een voorbeeld van hoe je de bibliotheek kunt gebruiken. In dit geval vinden we ook een voorbeeld met de naam *main.py*, maar het bevat veel



Figuur 7.13: De gedownloade bibliotheek op het Raspberry Pi Pico bord installeren.

meer informatie dan we nodig hebben (verbinding met de AHT21-sensor en verbinding met Wi-Fi). Open het bestand *main.py* en verwijder overbodige elementen, het programma zou er dan zo uit moeten zien:

```
importeer machine
importeer utime
importeer ENS160

#     instellen
print(" ENS160 instellen ")
i2c= machine. I2C (1 , sda= machine. Pin (14) , scl= machine . Pin
    →(15))
ens= ENS160 . ENS160 ( i2c)
ens. reset () utime
. sleep (0,5)
ens. operating_mode= 2 utime .
sleep (2,0)

terwijl True:
    #   take   reading   from   ENS160 print("
    Taking ENS160 measurements ... ") aqi= ens.
    AQI
    eco2= ens. CO2
    tvoc= ens. TVOC
    print(" AQI : "+ str( aqi)+ ", ECO2 : "+ str( eco2 )+ ",
        →TVOC : "+ str( tvoc))
```

```
21      utime . slaap (1)
```

Voer het programma uit en test het. Hoe interpreteer je de gelezen gegevens? Maak vertrouwd met de tabellen in de sensordocumentatie (zie Fig. 7.14, 7.15, 7.16).

AQI Reference

Level	Description	Suggestion	Recommended Stay Time
5	Extremely bad	In unavoidable circumstances, please strengthen ventilation	Not recommended to stay
4	Bad	Strengthen ventilation, find sources of pollution	Less than one month
3	Moderate	Strengthen ventilation, close to the water source	Less than 12 months
2	Good	Maintain adequate ventilation	Suitable for long-term living
1	Excellent	No suggestion	Suitable for long-term living

Figuur 7.14: AQI referentie.

eCO2/CO2 Concentration Reference

eCO2/CO2	Level	Result/Suggestion
21500	Terrible	Indoor air pollution is serious/Ventilation is required
1000-1500	Bad	Indoor air is polluted/ Ventilation is recommended
800-1000	Moderate	It is OK to ventilate
600-800	Good	Keep normal
400-600	Eexcellent	No suggestion

Afbeelding 7.15: Referentie eC02-concentratie.

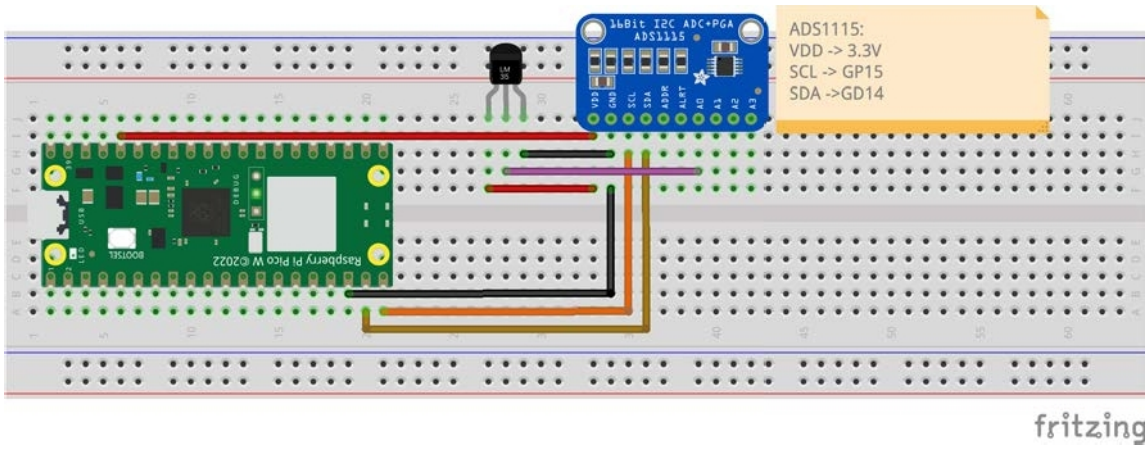
TVOC Concentration Reference

TOVC (ppb)	Effects on Human Health
>6000	Headaches and nerve problem
750-6000	Restless and headache
50-750	Restless and uncomfortable
<50	No effect

Afbeelding 7.16: TVOC-referentie.

7.5 Voorbeeld 11: Temperatuurmeting met externe A/D-omzetter

In dit voorbeeld leren we hoe we de spanningswaarde van een analoog-digitaalomzetter (ADC) kunnen lezen. Tot nu toe hebben we de ingebouwde analoog-digitaalomzetter gebruikt, die 12 bit is (resolutie van ongeveer 800 μV). Als we een betere nauwkeurigheid willen, kunnen we een externe converter gebruiken, bijvoorbeeld ADS1115 (16 bit - resolutie van ongeveer 76 μV). In dit voorbeeld sluiten we de LM35 temperatuursensor aan op de ADS1115 converter volgens Fig. 7.17.



Afbeelding 7.17: Het aansluiten van het elektronische circuit uit voorbeeld 11.

In het geval van de ADS1115 converter gebruiken we geen kant-en-klare bibliotheek die op Github te vinden is, maar schrijven we onze eigen bibliotheek. Waarom? De laatste twee voorbeelden lieten zien hoe je de SPI- en I2C-bussen kunt gebruiken met kant-en-klare bibliotheken en we hebben ons niet verdiept in het datatransmissieprotocol. Nu zullen we geen kant-en-klare bibliotheken gebruiken om te laten zien hoe je kunt communiceren met een converter of een andere sensor met een generieke communicatie-interface bibliotheek (die out of the box beschikbaar is in MicroPython voor Raspberry Pi Pico). Om te beginnen moet je een datasheet voor de ADS1115 converter op internet vinden (https://www.ti.com/lit/ds/syml ink/ads1115.pdf).
Dan moet je drie dingen vinden in de documentatie:

- schrijfprotocol
- leesprotocol
- registeradressen samen met de waarden die er naar toe moeten worden gestuurd om de gegeven parameters.

Dit is een algemeen schema van wat je moet doen, en in meer detail:

a. Scan welke apparaten zijn aangesloten op de I2C-bus om het adres van de A/D-omzetter te vinden. Om dit te doen configureer je eerst de I2C bussen met de machine.I2C() functie, die het I2C busnummer als eerste argument neemt (0 voor I2C0 of 1 voor I2C1 - de nummers staan in Fig. 1.1. In ons geval zal het I2C1 zijn). Het tweede argument is de klokfrequentie van de I2C bus. Het derde en vierde argument zijn de pinnen die zijn gebruikt om SDA en SCL aan te sluiten.

```
importeer machine
importeer utime

i2c= machine. I2C (1 , freq =400000 , scl= machine. Pin (15) ,
    →sda= machine . Pin (14))
```

b. Vervolgens gebruiken we de functie scan(), die de adressen van de apparaten die zijn aangesloten op de I2C-bus in de vorm van een lijst teruggeeft. Vervolgens lezen we de waarden uit de lijst met behulp van een for-lus. Om de gegevens in hexadecimale code weer te geven, wat de manier is waarop adressen meestal worden geschreven, gebruiken we de hexadecimale functie, die decimale waarden converteert naar hexadecimale:

```
importeer machine
importeer utime

i2c= machine. I2C (1 , freq =400000 , scl= machine. Pin (15) ,
    →sda= machine . Pin (14))
apparaten= i2c. scan ()
voor apparaat in apparaten:
            print( hex ( apparaat))
```

c. Voer de code uit en sla het leesadres op in de variabele ADS1115_I2C_ADDRESS:

```
importeer machine
importeer utime

i2c= machine. I2C (1 , freq =400000 , scl= machine. Pin (15) ,
    →sda= machine . Pin (14))
apparaten= i2c. scan ()
voor apparaat in apparaten:
     print( hex ( apparaat))

ADS 1115 _I2C_ADDRESS= 0 x48
```

Tip 7.5.1

Je kunt het verkregen adres van de ADS1115 A/D-omzetter controleren met tabel 7.2 van de ADS1115 datasheet. De hexadecimale waarde 0x48 is 0b1001000 in binair formaat. Je kunt: print(bin(device)) gebruiken in je programma.

d. Vervolgens moet je de adressen vinden van de registers in de ADC, waar de configuratie en de door de ADC gemeten waarden worden opgeslagen. De adressen zijn te vinden in de sensordocumentatie in Tabel 6, zoals getoond in Fig. 7.18. Merk op dat in deze tabel het register van de adreswijzer wordt beschreven met afzonderlijke bits. Een byte heeft 8 bits [7:0]. Bits van 7 (oudste) tot 2 zijn gereserveerd en moeten altijd met 0 worden geschreven. Bits van 1 tot 0 zijn belangrijk en we hebben 4 adressen, waarbij we alleen de adressen '00' en '01' gebruiken.

Geef de leesadressen door aan variabelen in het programma.

Table 6. Address Pointer Register Field Descriptions

Bit	Field	Type	Reset	Description
7:2	Reserved	W	0h	Always write 0h
1:0	P[1:0]	W	0h	**Register address pointer** 00 : Conversion register 01 : Config register 10 : Lo_thresh register 11 : Hi_thresh register

Afbeelding 7.18: Tabel 6 uit gegevensblad. Bron: https://www.ti.com/lit/ds/symlink/ads1115.pdf

```
importeer machine
importeer utime

i2c= machine. I2C (1 , freq =400000 , scl= machine. Pin (15) ,
    →sda= machine . Pin (14))
apparaten= i2c. scan ()
voor apparaat in apparaten:
            print( hex ( apparaat))

ADS 1115 _I2C_ADDRESS= 0 x48
ADS1115_CONVERSION_REG= 0 x00
ADS1115_CONFIG_REG= 0 x01
```

e. Vervolgens moet je informatie zoeken over welke parameters van de ADC kunnen worden ingesteld en welke waarden daarvoor naar het config-register moeten worden gestuurd. Al deze gegevens staan in Tabel 8 (zie Fig. 7.19 en 7.20) en de letter h-aanduidingen geven waarden aan die in hexadecimale code zijn geschreven, bijvoorbeeld 1h=0x01, die moeten worden ingesteld op individuele bits van het 16-bits configuratieregister. In dit voorbeeld voeren we een enkele meting uit van kanaal 0 in het bereik van± 4,096V. We zullen geen comparator gebruiken. De benodigde waarden voor de variabelen zijn opgeschreven:

```
import   machine
importeer utime

i2c= machine. I2C (1 , freq =400000 , scl= machine. Pin (15) ,
    →sda= machine . Pin (14))
apparaten= i2c. scan ()
voor apparaat in apparaten:
     print( hex ( apparaat))

ADS 1115 _I2C_ADDRESS= 0 x48
ADS1115_CONVERSION_REG= 0 x00
ADS1115_CONFIG_REG= 0 x01


ADS1115_CONFIG_OS_SINGLE= 0 x8000          #   Start een
    →conversie                             single
ADS1115_CONFIG_MUX_AIN0   =   0 x4000      # AIN0 Ketting ( Single -
    →eindigde kanaal)
ADS 1115 _CONFIG_GAIN= 0 x0200          # Versterking   versterker   + -
4.096 V
```

```
16 ADS1115_CONFIG_MODE_SINGLE  = 0x0100          # Modus enkel schot
17 ADS1115_CONFIG_DR_128 SPS= 0x0080             # Data snelheid = 128
       →SPS
18 ADS1115_CONFIG_COMP_DISABLED  = 0x0003        # uitschakelen
       →vergelijker
```

Table 8. Config Register Field Descriptions

Bit	Field	Type	Reset	Description
15	OS	R/W	1h	**Operational status or single-shot conversion start** This bit determines the operational status of the device. OS can only be written when in power-down state and has no effect when a conversion is ongoing. When writing: 0 : No effect 1 : Start a single conversion (when in power-down state) When reading: 0 : Device is currently performing a conversion 1 : Device is not currently performing a conversion
14:12	MUX[2:0]	R/W	0h	**Input multiplexer configuration (ADS1115 only)** These bits configure the input multiplexer. These bits serve no function on the ADS1113 and ADS1114. 000 : AIN_P = AIN0 and AIN_N = AIN1 (default) 001 : AIN_P = AIN0 and AIN_N = AIN3 010 : AIN_P = AIN1 and AIN_N = AIN3 011 : AIN_P = AIN2 and AIN_N = AIN3 100 : AIN_P = AIN0 and AIN_N = GND 101 : AIN_P = AIN1 and AIN_N = GND 110 : AIN_P = AIN2 and AIN_N = GND 111 : AIN_P = AIN3 and AIN_N = GND
11:9	PGA[2:0]	R/W	2h	**Programmable gain amplifier configuration** These bits set the FSR of the programmable gain amplifier. These bits serve no function on the ADS1113. 000 : FSR = ±6.144 V(1) 001 : FSR = ±4.096 V(1) 010 : FSR = ±2.048 V (default) 011 : FSR = ±1.024 V 100 : FSR = ±0.512 V 101 : FSR = ±0.256 V 110 : FSR = ±0.256 V 111 : FSR = ±0.256 V
8	MODE	R/W	1h	**Device operating mode** This bit controls the operating mode. 0 : Continuous-conversion mode 1 : Single-shot mode or power-down state (default)
7:5	DR[2:0]	R/W	4h	**Data rate** These bits control the data rate setting. 000 : 8 SPS 001 : 16 SPS 010 : 32 SPS 011 : 64 SPS 100 : 128 SPS (default) 101 : 250 SPS 110 : 475 SPS 111 : 860 SPS

Afbeelding 7.19: Beschrijving van de velden in het Config-register - deel 1. Bron: https://www.ti.com/lit/ds/symlink/ads1115.pdf

Table 8. Config Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
4	COMP_MODE	R/W	0h	**Comparator mode (ADS1114 and ADS1115 only)** This bit configures the comparator operating mode. This bit serves no function on the ADS1113. 0 : Traditional comparator (default) 1 : Window comparator
3	COMP_POL	R/W	0h	**Comparator polarity (ADS1114 and ADS1115 only)** This bit controls the polarity of the ALERT/RDY pin. This bit serves no function on the ADS1113. 0 : Active low (default) 1 : Active high
2	COMP_LAT	R/W	0h	**Latching comparator (ADS1114 and ADS1115 only)** This bit controls whether the ALERT/RDY pin latches after being asserted or clears after conversions are within the margin of the upper and lower threshold values. This bit serves no function on the ADS1113. 0 : Nonlatching comparator . The ALERT/RDY pin does not latch when asserted (default). 1 : Latching comparator. The asserted ALERT/RDY pin remains latched until conversion data are read by the master or an appropriate SMBus alert response is sent by the master. The device responds with its address, and it is the lowest address currently asserting the ALERT/RDY bus line.
1:0	COMP_QUE[1:0]	R/W	3h	**Comparator queue and disable (ADS1114 and ADS1115 only)** These bits perform two functions. When set to 11, the comparator is disabled and the ALERT/RDY pin is set to a high-impedance state. When set to any other value, the ALERT/RDY pin and the comparator function are enabled, and the set value determines the number of successive conversions exceeding the upper or lower threshold required before asserting the ALERT/RDY pin. These bits serve no function on the ADS1113. 00 : Assert after one conversion 01 : Assert after two conversions 10 : Assert after four conversions 11 : Disable comparator and set ALERT/RDY pin to high-impedance (default)

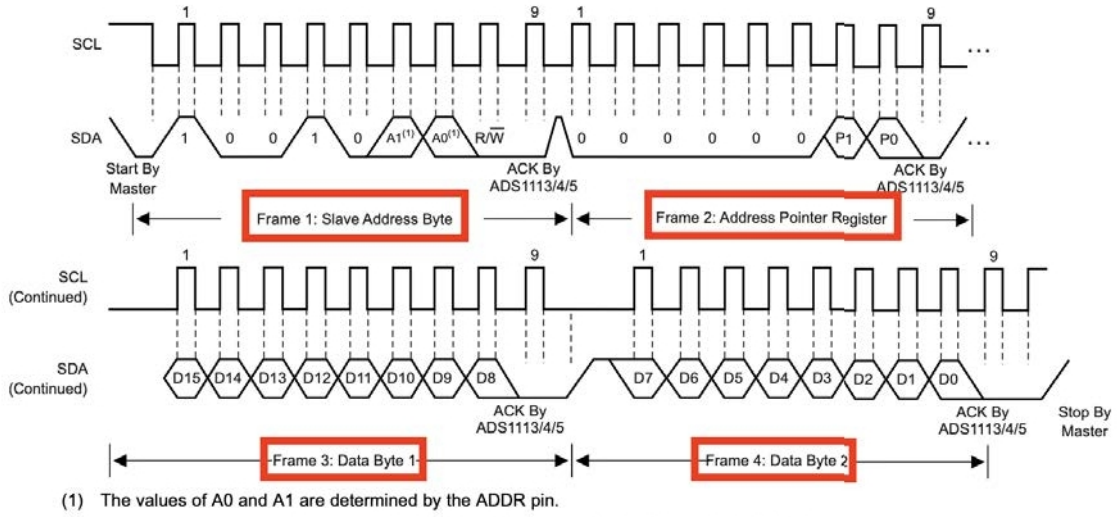
Afbeelding 7.20: Beschrijving van de velden in het Config-register - deel 2. Bron: https://www.ti.com/lit/ds/symlink/ads1115.pdf

f. In de volgende stap combineren we alle ingestelde bits tot één 16-bits waarde om deze naar het configuratieregister te sturen:

```
importeer machine
importeer utime

i2c= machine. I2C (1 , freq =400000 , scl= machine. Pin (15) ,
    →sda= machine . Pin (14))
apparaten= i2c. scan ()
voor apparaat in apparaten:
      print( hex ( apparaat))

ADS 1115 _I2C_ADDRESS= 0 x48
ADS1115_CONVERSION_REG= 0 x00
ADS1115_CONFIG_REG= 0 x01

ADS1115_CONFIG_OS_SINGLE= 0 x8000
ADS1115_CONFIG_MUX_AIN0= 0 x4000 ADS
1115 _CONFIG_GAIN= 0 x0200
ADS1115_CONFIG_MODE_SINGLE= 0 x0100 ADS1115_CONFIG_DR_128
SPS= 0 x0080
ADS1115_CONFIG_COMP_DISABLED  = 0 x0003

ADS1115_CONFIG=(
ADS1115_CONFIG_OS_SINGLE
| ADS1115 _CONFIG_MUX_AIN0
```

```
| ADS 1115 _CONFIG_GAIN
| ADS 1115 _CONFIG_MODE_SINGLE
|  ADS1115_CONFIG_DR_128 SPS
| ADS1115 _CONFIG_COMP_DISABLED
)
```

g. In de volgende stap zullen we de functie maken om onze waarde naar het register te sturen. Om dit te doen, moeten we kijken hoe het communicatieprotocol eruit ziet. Volgens de documentatie (zie fig. 7.21), moet je eerst het apparaatadres (ADC I2C adres) sturen, dan het registeradres (bijv. configuratie) en dan de databytes. De ADS1115-converter is een 16-bits apparaat, en je moet eerst de 8 oudste bits verzenden (aanduiding: D15-D8) en dan de 8 jongste bits (aanduiding: D7-D0). Dit wordt Big-endian genoemd. Gewoonlijk worden waarden in ons computergeheugen opgeslagen in Little-endian configuratie, wat het tegenovergestelde is van dit. We zullen te maken krijgen. Het zal het handigst zijn om een functie te maken voor het verzenden van gegevens en daarbinnen een array van 4 elementen te maken met de gegevens die verzonden moeten worden (apparaatadres, registeradres, oudste bits, jongste bits). In de array slaan we gegevens op in de vorm van bytes, vandaar dat we de functie bytearray() gebruiken, die een array maakt met gegevens die zijn opgeslagen in de vorm van bytes.



(1) The values of A0 and A1 are determined by the ADDR pin.

Figure 31. Timing Diagram for Writing to ADS111x

Afbeelding 7.21: Timing diagram van schrijven. Bron: https://www.ti.com/lit/ds/symlink/ads1115.pdf

```
import  machine
---

ADS1115_CONFIG=(
            ADS1115_CONFIG_OS_SINGLE
            | ADS1115 _CONFIG_MUX_AIN0
            | ADS 1115 _CONFIG_GAIN
            | ADS1115 _CONFIG_MODE_SINGLE
            |  ADS1115_CONFIG_DR_128 SPS
            | ADS1115 _CONFIG_COMP_DISABLED
```

```
)

def  write_ads1115_register( register ,  waarde):
    data= bytearray ([ register , ( waarde>> 8) & 0 xFF ,
        →waarde & 0 xFF ])
```

In dit fragment hierboven, wanneer we een bytearray maken, zie je misschien twee onbekende ingangen:

- (waarde>> 8) - dit is een bewerking waarbij 8 bits naar rechts worden verschoven, dus als de gegevens 16 bits zijn, bijvoorbeeld 1010 1010 0011 0011, zal het resultaat 0000 0000 zijn 1010 1010. Dus de 8 oudere bits springen naar de plaats van de 8 jongere bits, en in plaats daarvan zouden er alleen nullen moeten staan.
- waarde & 0xFF - dit is een logische AND bewerking, waarvan het resultaat zal zijn een ongewijzigde 16-bits waarde. Waarom zo'n bewerking? Als de waardevariabele iets meer bevatte dan alleen de 16 bits die ons interesseren, dan verwijderen we de rest van de informatie en geven we alleen 16 bits door. In Pythoneen dynamische taal, weten we niet zeker welk type waarde is doorgegeven. Dit kan een 32-bits waarde zijn met enkele hogere bits ingesteld en een shift-operatie kan bits naar een 16-bits veld brengen. Dus om het zekere voor het onzekere te nemen, is het beter om de waarde goed te maskeren met een AND booleaanse bewerking.

h. De volgende stap is het verzenden van gegevens van de tabel naar de ADC via de I2C-bus:

```
import  machine
---

ADS1115_CONFIG=(
          ADS1115_CONFIG_OS_SINGLE
          | ADS1115 _CONFIG_MUX_AIN0
          | ADS 1115 _CONFIG_GAIN
          | ADS1115 _CONFIG_MODE_SINGLE
          |  ADS1115_CONFIG_DR_128 SPS
          | ADS1115 _CONFIG_COMP_DISABLED
)

def  write_ads1115_register( register ,  waarde):
    data= bytearray ([ register , ( waarde>> 8) & 0 xFF ,
        →waarde & 0 xFF ])
    i2c. writeto ( ADS1115_I2C_ADDRESS ,  data)
```

i. We weten al hoe we gegevens moeten verzenden. Het is tijd om een functie te maken voor het lezen van gegevens. Om dit te doen, moeten we informatie vinden over het protocol voor het lezen van gegevens in de datasheet (zie Fig. 7.22). Het lezen van gegevens gebeurt in twee stappen. De eerste stap is het verzenden van het ADC I2C-adres en het registeradres (door het instellen van het Address Pointer Register) waarvan we data willen lezen (rode rechthoeken in Fig. 7.22). De tweede stap is het lezen van twee bytes data van het ADC adres (donkerblauwe rechthoeken in Fig. 7.22). Laten we dus een functie maken die gegevens uit het register leest:

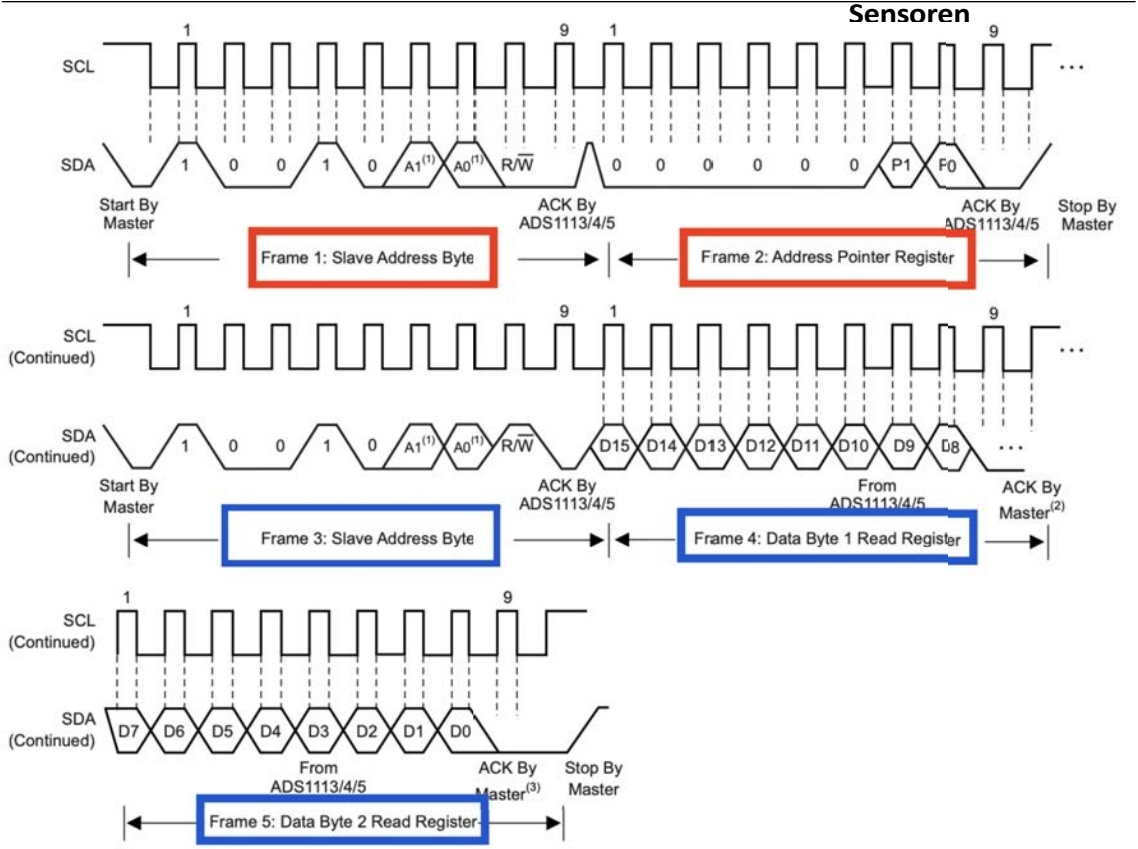


Figure 30. Timing Diagram for Reading From ADS111x

Afbeelding 7.22: Timingdiagram van lezen. Bron: https://www.ti.com/lit/ds/sy mlink/ads1115.pdf

```
import machine
---

ADS1115_CONFIG=(
            ADS1115_CONFIG_OS_SINGLE
            | ADS1115 _CONFIG_MUX_AIN0
            | ADS 1115 _CONFIG_GAIN
            | ADS1115 _CONFIG_MODE_SINGLE
            |  ADS1115_CONFIG_DR_128 SPS
            | ADS1115 _CONFIG_COMP_DISABLED
)

def write_ads1115_register( register , waarde):
    data= bytearray ([ register , ( waarde>> 8) & 0 xFF ,
        →waarde & 0 xFF ])
    i2c. writeto ( ADS1115_I2C_ADDRESS , data)

def read_ads1115_register( register , num_bytes):

```

```
18      i2c. writeto ( ADS1115_I2C_ADDRESS ,    bytearray ([
            →register ])
19      return i2c. readfrom ( ADS1115 _I2C_ADDRESS , num_bytes
            →)
```

j. We hebben al functies voor het lezen en schrijven van gegevens via de I2C-bus. Laten we nu een functie maken waarin we de AD-converter configureren en er gegevens van lezen en deze vervolgens omzetten naar spanning. Hier zullen we de functie *from_bytes()* gebruiken, die waarden converteert van individuele bytes naar een decimale waarde. Merk op dat we in *de from_bytes()* functie moeten specificeren hoe de bytes in het geheugen zijn gerangschikt. Omdat deze werden gelezen als Big-endian, gebruiken we **'big'** specificatie.

```
1  import   machine
2  ...
3
4  def  read_ads1115_register( register ,  num_bytes):
5       i2c. writeto ( ADS1115_I2C_ADDRESS ,  bytearray ([
            →register ])
6       return i2c. readfrom ( ADS1115 _I2C_ADDRESS , num_bytes
            →)
7
8  def read_adc ():
9       # Verzenden van  configuratie  naar  configuratie  register
10      write_ads1115_register( ADS 1115 _CONFIG_REG ,
            →ADS1115_CONFIG )
11
12      # Wacht tot de meting klaar is
13      utime . sleep (0,01)
14
15      # Inlezen  conversie  waarde
16      resultaat= read_ads1115 _register(
            '→ ADS1115_CONVERSION_REG , 2)
17      raw_value= int. from_bytes ( resultaat , ' groot')
18      indien raw_value>= 0 x8000 :    # Correctie  voor  negatief
            →nummers
19          raw_value -  = 0 x10000
20
21       # Waarde omzetten naar spanning
22       spanning  =  raw_value *    (4,096 /    32768)  # Bereik
            →+-4,096 V/ max waarde
23       retourspanning
```

k. Laten we nu de hoofdlus toevoegen waarin we de spanningswaarde lezen en omzetten naar temperatuur (vermenigvuldiging met 100 is volgens de documentatie voor de LM35 temperatuursensor):

```
1  importeer machine
2  importeer utime
3
4  i2c= machine. I2C (1 , freq =400000 , scl= machine. Pin (15) ,
       →sda= machine . Pin (14))
5  apparaten= i2c. scan ()
```

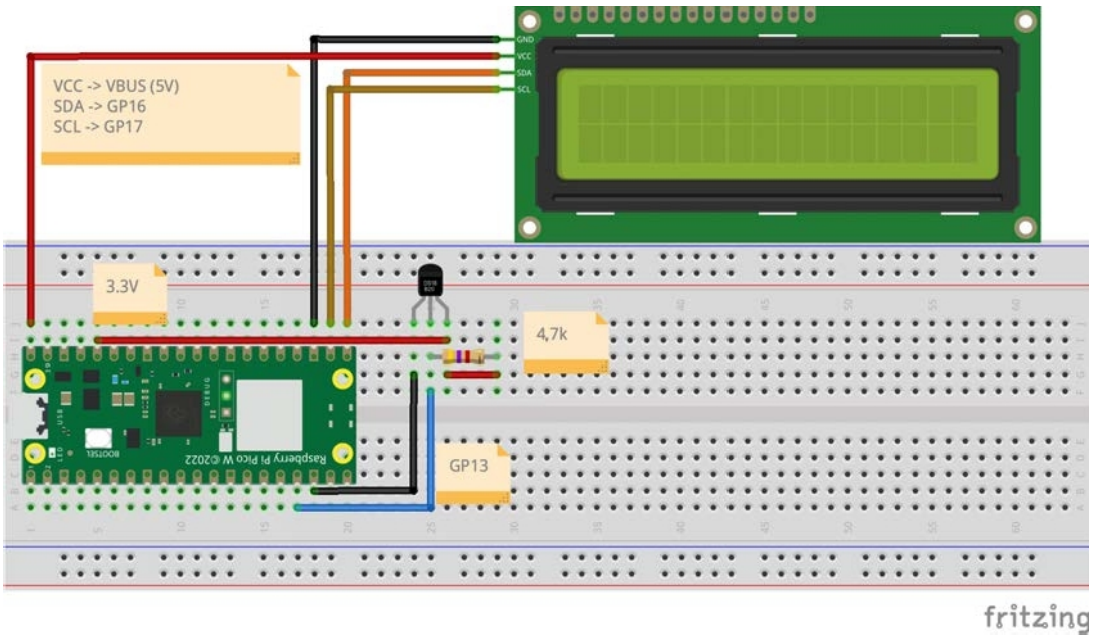
```
6  voor apparaat in apparaten:
7      print( hex ( apparaat))
8
9  ADS 1115 _I2C_ADDRESS= 0 x48
10 ADS1115_CONVERSION_REG= 0 x00
11 ADS1115_CONFIG_REG= 0 x01
12
13 ADS1115_CONFIG_OS_SINGLE= 0 x8000
14 ADS1115_CONFIG_MUX_AIN0 = 0 x4000
15 ADS 1115 _CONFIG_GAIN= 0 x0200
16 ADS1115_CONFIG_MODE_SINGLE = 0 x0100
17 ADS1115_CONFIG_DR_128 SPS= 0 x0080
18 ADS1115_CONFIG_COMP_DISABLED = 0 x0003
19
20 ADS1115_CONFIG=(
21             ADS1115_CONFIG_OS_SINGLE
22             | ADS1115 _CONFIG_MUX_AIN0
23             | ADS 1115 _CONFIG_GAIN
24             | ADS1115 _CONFIG_MODE_SINGLE
25             |  ADS1115_CONFIG_DR_128 SPS
26             | ADS1115 _CONFIG_COMP_DISABLED
27 )
28
29 def write_ads1115_register( register , waarde):
30     data= bytearray ([ register , ( waarde>> 8) & 0 xFF ,
           →waarde & 0 xFF ])
31     i2c. schrijf naar ( ADS1115_I2C_ADDRESS , gegevens)
32
33 def read_ads1115_register( register , num_bytes):
34     i2c. writeto ( ADS1115_I2C_ADDRESS , bytearray ([
           →register ])
35     return i2c. readfrom ( ADS1115 _I2C_ADDRESS , num_bytes
           →)
36
37 def read_adc ():
38     # Verzenden van configuratie naar configuratie register
39     write_ads1115_register( ADS 1115 _CONFIG_REG ,
           →ADS1115_CONFIG )
40
41     # Wacht tot de meting klaar is
42     utime . slaap (0,01)
43
44     # Inlezen conversie waarde
45     resultaat= read_ads1115 _register(
           '→ ADS1115_CONVERSION_REG , 2)
46     raw_value= int. from_bytes ( resultaat , ' groot')
47     if raw_value>= 0 x8000 : # Correctie voor negatief
           →nummers
48         raw_value - = 0 x10000
49
```

```
50      # Waarde omzetten naar spanning
51      spanning = raw_value * (4,096 / 32768) # Bereik
            →+-4,096 V/ max. waarde
52      retourspanning
53
54  terwijl True:
55      spanning= read_adc ()
56      temp = spanning *100
57      print("T="+ str( temp ))
58      utime . slaap (1)
```

De code is klaar. Test hem uit.

7.6 Voorbeeld 12: Temperatuurmeting met 1-draads bus

De 1-draads interface is ontwikkeld door het bedrijf Dallas Semiconductor. Deze bus bestaat uit slechts één lijn. Deze interface werkt hetzelfde als de I2C-bus, maar de 0- en 1-bits worden gedefinieerd door de tijdsduur van het lage signaal. De 1-draads bus werkt langzamer dan I2C. De maximale snelheid is 16 kbit/s.
In dit voorbeeld lezen we de temperatuur uit van de DS18B20 temperatuursensor via de 1-draads bus en geven we de uitgelezen temperatuur weer op een LCD-display met een I2C-converter (soms wordt de converter apart aangeschaft voor het display). Sluit het systeem aan zoals in Fig. 7.23.



Afbeelding 7.23: Het elektronische circuit van voorbeeld 12 aansluiten.

Laten we nu een programma schrijven dat gegevens van de DS18B20-sensor leest en weergeeft op het scherm. Volg deze stappen om dit te doen:

a. Installeer de onwire bibliotheek en ds18x20 met behulp van de pakketbeheerder in de Thonny editor (zie Fig. 7.24 en 7.25).

Afbeelding 7.24: Installatie van de onewire bibliotheek.

Afbeelding 7.25: Installatie van de DS18x20 bibliotheek.

b. Laten we nu de benodigde bibliotheken toevoegen:

```
importeer machine
importeer onewire
importeer ds18x20
importeer time
```

c. Vervolgens moet je opgeven welke GPIO-pin zal dienen als de 1-draads bus. Gebruik hiervoor de functie *onewire.OneWire(pinnummer)*. De volgende stap is het configureren van de DS18B20-sensor, d.w.z. het aanroepen van de functie: *ds18x20.DS18X20()*, die een onewire-klasseobject als argument neemt:

```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime

one_wire_bus= onewire. OneWire( machine. Pin (13))
ds18b20_sensor= ds18x20 . DS18X 20 ( one_wire_bus)
```

d. In de volgende stap moet je het adres bepalen van de DS18B20 sensor die we op de 1-draads bus hebben aangesloten. Dit kan automatisch gebeuren met de functie scan():

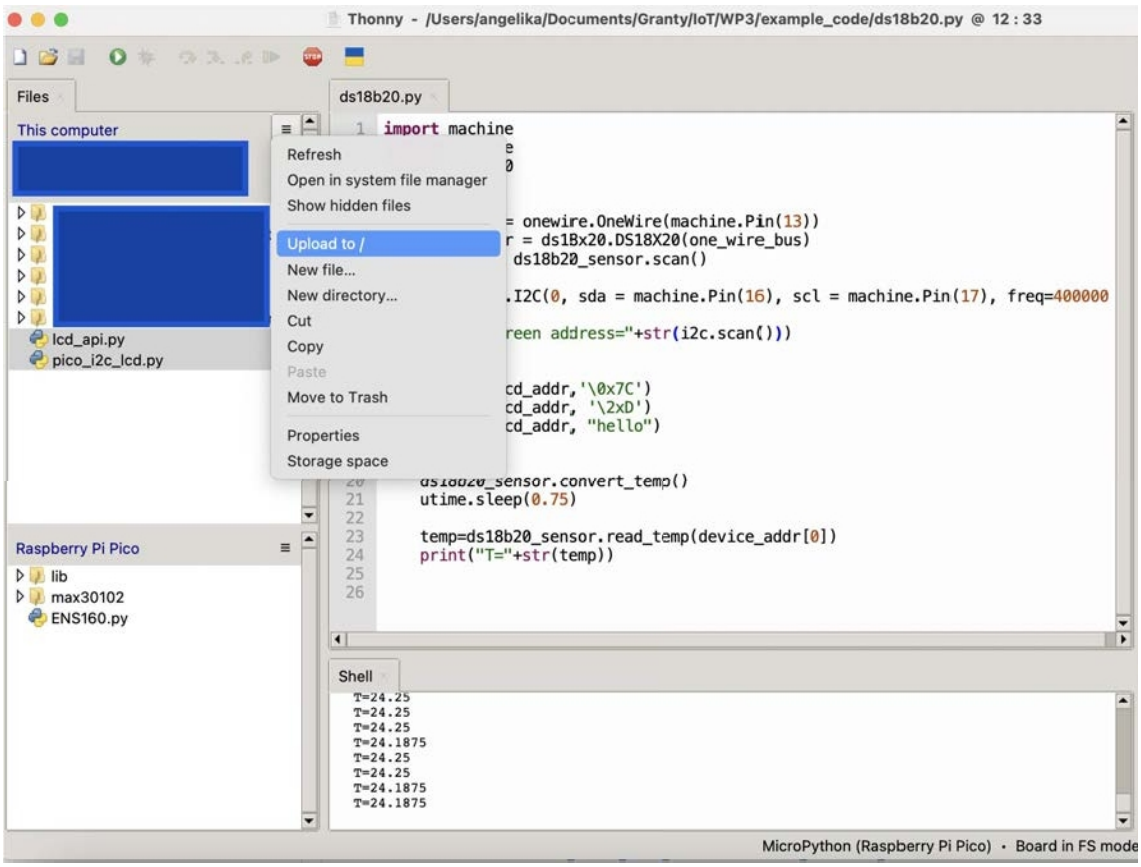
```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime

one_wire_bus= onewire. OneWire( machine. Pin (13))
ds18b20_sensor= ds18x20 . DS18X 20 ( one_wire_bus) device_addr=
ds18 b 20 _sensor. scan ()
```

e. Vervolgens moet je de sensor een commando geven om de temperatuur te meten (*con- vert_temp()* functie) en dan ongeveer 750 ms wachten tot hij meet volgens de documentatie voor de DS18B20 sensor. Vervolgens moet je de gemeten temperatuurwaarde in Celsius lezen met de functie *read_temp()*, die het adres van de sensor als argument neemt. Aangezien de *scan()* functie een array met adressen retourneert, moet je het eerste element uit de array halen. Arrays in Python zijn genummerd vanaf 0 en om een bepaald fragment van de array te extraheren, moet je *array_name[index]* invoeren:

```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime

one_wire_bus= onewire. OneWire( machine. Pin (13))
ds18b20_sensor= ds18x20 . DS18X 20 ( one_wire_bus) device_addr=
ds18 b 20 _sensor. scan ()

terwijl True:
    ds18b20_sensor. convert_temp ()
    utime . sleep (0,75)

    temp= ds18 b 20 _sensor. read_temp ( device_addr [0])
    print("T=" + str( temp ))
```

f. Je kunt nu het programma testen. Als de temperatuur correct wordt afgelezen, kunnen we verder gaan met het LCD-scherm. Download eerst twee bestanden: *lcd_api.py* (https://github.com/T-622/RPI-PICO-I2C-LCD/blob/main/lcd_api.py) en *i2c_lcd.py* (https://github.com/T-622/RPI-PICO-I2C-LCD/blob/main/pico_i2c_lcd.py) en upload deze naar de Raspbery Pi Pico zoals weergegeven in

Fig. 7.26.



Afbeelding 7.26: Bibliotheken voor het LCD-scherm installeren

g. Laten we nu de benodigde bibliotheken toevoegen:

```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime
van lcd_api importeer
uit pico_i2c_lcd importeer

one_wire_bus= onewire. OneWire( machine. Pin (13))
ds18b20_sensor= ds18x20 . DS18X 20 ( one_wire_bus) device_addr=
ds18 b 20 . sensor. scan ()
```

h. Laten we vervolgens de I2C-bus configureren en scannen om het adres van het LCD-scherm te vinden. Start het programma en lees de waarde.

```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime
van lcd_api importeer LcdApi
uit pico_i2c_lcd importeer I2cLcd
```

```
7
8  one_wire_bus= onewire. OneWire( machine. Pin (13))
9  ds18b20_sensor= ds18x20 . DS18X 20 ( one_wire_bus) device_addr=
10 ds18 b 20 _sensor. scan ()
11
12 i2c= machine. I2C (0 , sda= machine . Pin (16) , =
       →machine. Pin (17),    freq= 400000) print("
13 LCD screen address="+ str( i2c. scan ())
14 ...
```

i. Laten we het adres van het gelezen LCD-scherm opslaan in een variabele en variabelen maken om de afmetingen van het scherm (aantal rijen en kolommen) op te slaan:

```
1  importeer
2  machine
3  importeer
4  onewire
5  importeer
6  ds18x20
7  importeer utime
8  van lcd_api importeer LcdApi
9  uit pico_i2c_lcd importeer I2cLcd
10
11 one_wire_bus= onewire. OneWire( machine. Pin (13))
12 ds18b20_sensor= ds18x20 . DS18X 20 ( one_wire_bus) device_addr=
   ds18 b 20 _sensor. scan ()
13
14 i2c= machine. I2C (0 , sda= machine . Pin (16) , =
       →machine. Pin (17),    freq= 400000) print("
15 LCD screen address="+ str( i2c. scan ())
16
17 I2C_ADDR =63
18 I2C_NUM_ROWS=2
   I2C_NUM_COLS=16
```

j. Laten we het LCD-scherm initialiseren en 1s wachten. Vervolgens wissen we het scherm, plaatsen we de cursor aan het begin en geven we de welkomsttekst weer:

```
1  importeer
2  machine
3  importeer
4  onewire
5  importeer
6  ds18x20
7  importeer utime
8  van lcd_api importeer LcdApi
9  uit pico_i2c_lcd importeer I2cLcd
10
11 one_wire_bus= onewire. OneWire( machine. Pin (13))
12 ds18b20_sensor= ds18x20 . DS18X 20 ( one_wire_bus) device_addr=
   ds18 b 20 _sensor. scan ()
13
14 i2c= machine. I2C (0 , sda= machine . Pin (16) , =
       →machine. Pin (17),    freq= 400000) print("
15 LCD screen address="+ str( i2c. scan ())

   I2C_ADDR =63
```

```
I2C_NUM_ROWS=2
I2C_NUM_COLS=16

lcd= I2cLcd ( i2c , I2C_ADDR , I2C_NUM_ROWS , I2 C_NUM_COLS
    '→     ) utime .
sleep (1)

lcd.    clear    () lcd.
move_to (0,0) lcd.
putstr(" Hello")
...
```

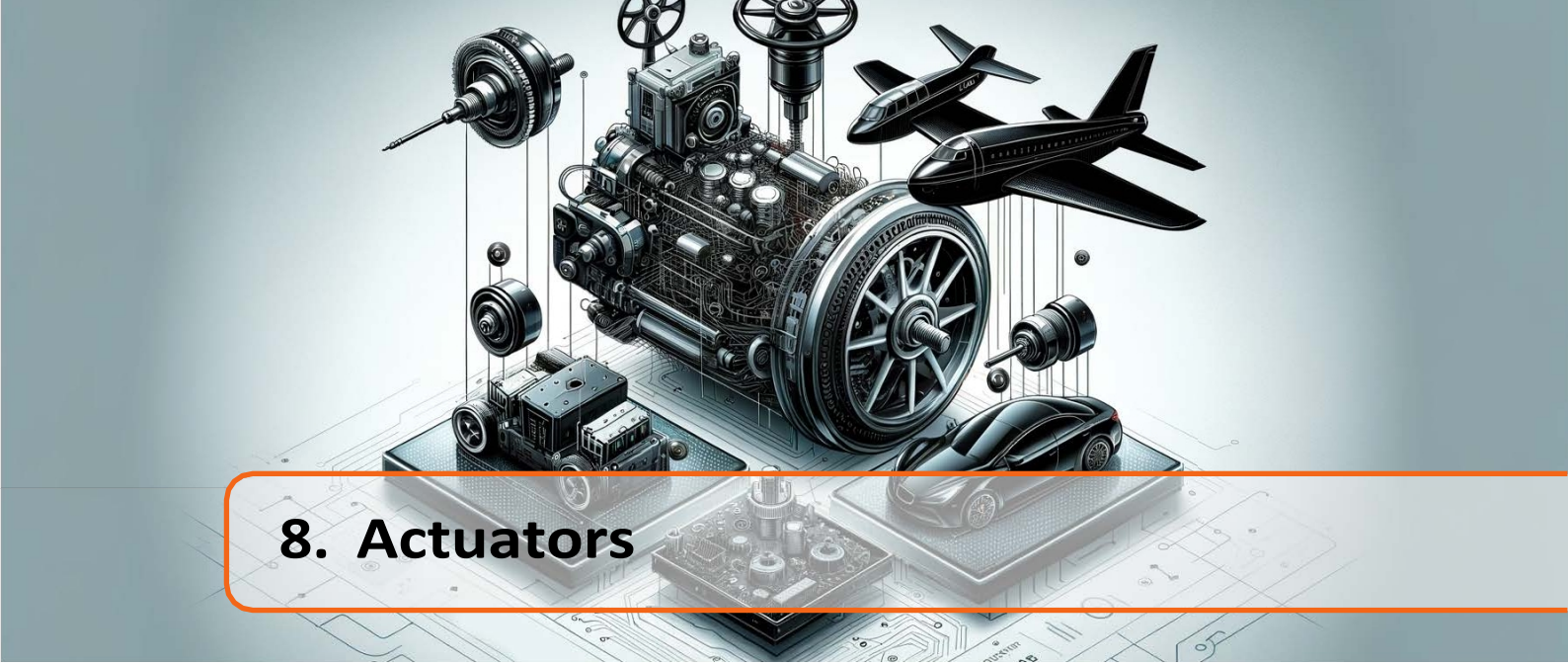
k. De laatste stap is het weergeven van de temperatuur op de volgende rij van het LCD-scherm nadat de temperatuur gemeten is:

```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime
van lcd_api importeer LcdApi
uit pico_i2c_lcd importeer I2cLcd

one_wire_bus= onewire. OneWire( machine. Pin (13))
ds18b20_sensor= ds18x20 . DS18X 20 ( one_wire_bus) device_addr=
ds18 b 20 _sensor. scan ()

i2c= machine. I2C (0 , sda= machine . Pin (16) , =
    →machine. Pin (17),    freq= 400000) print("
LCD screen address="+ str( i2c. scan ())

I2C_ADDR =63
I2C_NUM_ROWS=2
I2C_NUM_COLS=16

lcd= I2cLcd ( i2c , I2C_ADDR , I2C_NUM_ROWS , I2 C_NUM_COLS
    '→     ) utime .
sleep (1)

lcd.    clear    () lcd.
move_to (0,0) lcd.
putstr(" Hello")

terwijl True:
      ds18b20_sensor. convert_temp ()
      utime . sleep (0,75)

      temp= ds18 b 20 _sensor. read_temp ( device_addr [0])
      print("T="+ str( temp ))

      lcd.      move_to      (0      ,1) lcd.
      putstr("T="+ str( temp )+" C")
```

Het programma is klaar en kan worden getest.

Tip 7.6.1

Als je het LCD-scherm wilt gebruiken zonder een I2C-converter, kun je de bibliotheek gebruiken die hier beschikbaar is: https://github.com/gusandrio li/liquid-crystal-pico.

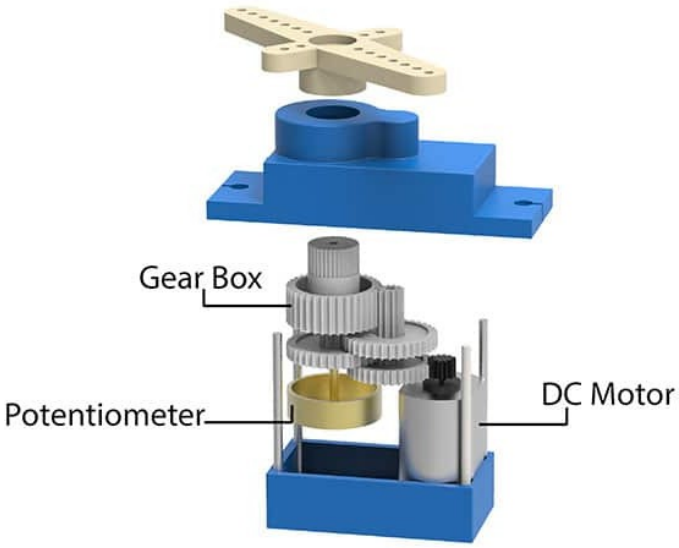


8. Actuators

Actuatoren zijn elementen die een beweging maken (uitvoerende elementen). De basis actuatoren zijn: servo, stappenmotor en gelijkstroommotor.

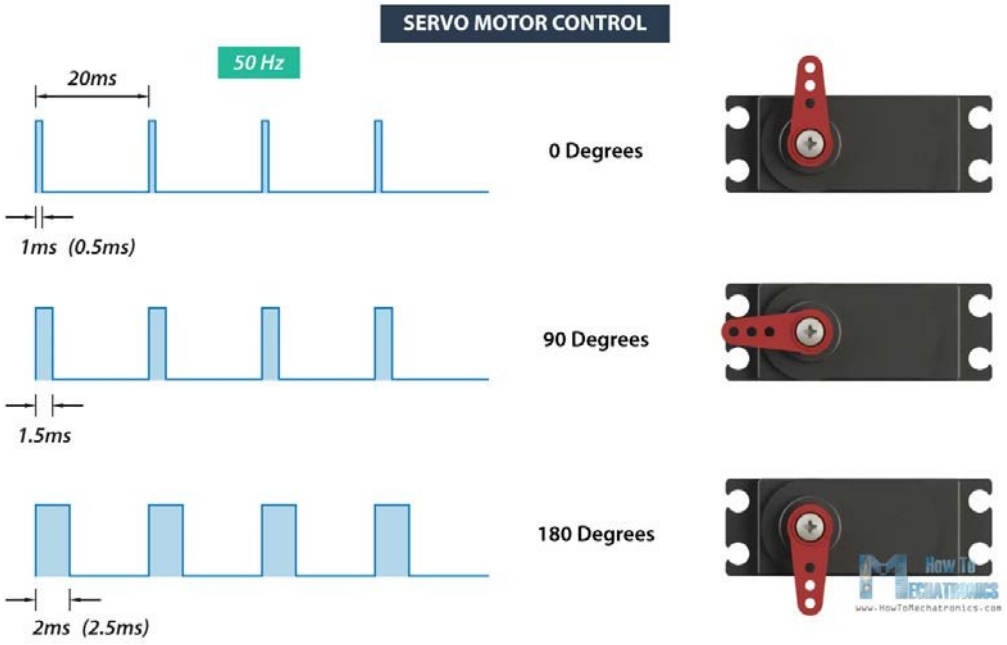
8.1 Voorbeeld 13: Servo motor

De servo is een motor met een tandwiel dat onder een bepaalde hoek draait, meestal in het bereik van $(0;180)^o$ of $(-90;90)^o$. De constructie van de servo wordt getoond in Fig. 8.1. De servo kan rechtstreeks op het Raspberry Pi Pico-bord worden aangesloten. De drie draden komen uit de servo. De middelste (meestal rood) moet worden aangesloten op de voeding (+5V = VBUS). De zwarte is GND en de laatste draad (meestal licht gekleurd: wit/geel) is de aansturing die verbonden moet worden met de PWM-pin.



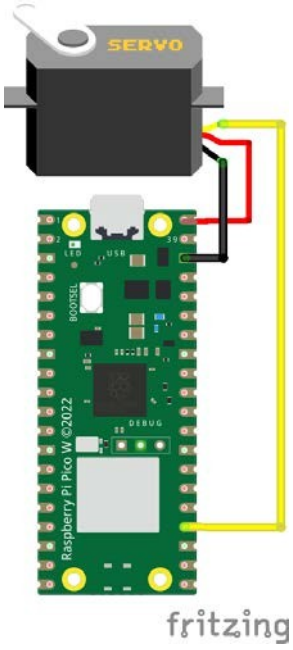
Afbeelding 8.1: Werkingsprincipe van een servomechanisme. Bron: https://ai.thestempedia.com/docs/quarky/quarky-technical-specifications/servomotor-with-quarky/.

De servobesturing is gebaseerd op het PWM-signaal. De controller in de servo leest het PWM-signaal en bepaalt op basis daarvan de hoek waaronder het tandwiel moet draaien. Het idee van de servobesturing wordt getoond in Fig. 8.2.



Afbeelding 8.2: Principe van servobesturing. Bron: https://howtomechatronics.com/wp-content/uploads/2018/03/RC-Servomotorbesturing-signaal.png.

Laten we een programma maken dat de servo van de minimum naar de middenpositie draait en dan naar maximum en terug. Om dit te doen, sluiten we eerst het systeem aan zoals getoond in Fig. 8.3 (rode draad - VBUS, gele draad - GP18, zwarte draad - GND).



Afbeelding 8.3: Het elektronische circuit van voorbeeld 13 aansluiten.

Open de Thonny editor en volg deze stappen:

1. Voeg de benodigde bibliotheken toe:

```
importeer machine
importeer utime
```

2. Volgens de documentatie van de servo is de minimumpositie wanneer het hoogsignaal 0,9 ms duurt (soms 0,5 ms). De middelste stand is als het hoogsignaal 1,5 ms duurt en de maximale stand is 2,1 ms (soms 2,5 ms). De PWM-signaalperiode moet 20 ms (50 Hz) zijn. Laten we variabelen maken die deze waarden opslaan in ns:

```
importeer machine
importeer utime

MIN= 900000
MID= 000
MAX= 2100000
```

3. Configureer de GP18-pin als PWM en stel de PWM-signaalfrequentie in op 50 Hz.

```
importeer machine
importeer utime

MIN= 900000
MID= 000
MAX= 2100000

pwm= machine. PWM ( machine . Pin (18))
pwm . freq (50)
```

4. We zullen de servoposities definiëren door de duty cycle in te stellen. In het vorige hoofdstuk werd de functie *duty_u16()* gepresenteerd, die de duty cycle instelt in het bereik van 0 tot 65535 (100%). Het zal hier handiger zijn om de functie *duty_ns()* te gebruiken, die de duty cycle voor een opgegeven tijd in nanoseconden instelt. Laten we de initiële afbuiging instellen op het midden:

```
importeer machine
importeer utime

MIN= 900000
MID= 000
MAX= 2100000

pwm= machine. PWM ( machine . Pin (18))
pwm . freq (50)
pwm . duty_ns( MIN )
```

5. In de hoofdlus zorgen we ervoor dat de servo elke 1s van de minimumpositie door het midden naar het maximum en terug draait:

```
importeer machine
importeer utime

```

```
MIN= 900000
MID= 000
MAX= 2100000

pwm= machine. PWM ( machine . Pin (18))
pwm . freq (50)
pwm . duty_ns( MIN )

terwijl True:
      pwm . duty_ns( MIN )
      utime . slaap (1)
      pwm . duty_ns( MID )
      utime . slaap (1)
      pwm . duty_ns( MAX )
      utime . slaap (1)
      pwm . duty_ns( MID )
      utime . slaap (1)
```

Het programma is klaar, test de werking.

8.2 Voorbeeld 14: Slimme ventilatie

In dit voorbeeld maken we een slim ventilatiesysteem dat bestaat uit een Pololu DS18B20 temperatuursensor (uit hoofdstuk 7.6) en een gelijkstroommotor die via een DRV8835 regelaar is aangesloten op de Raspberry Pi Pico. Afhankelijk van de temperatuur zullen we de snelheid van de gelijkstroommotor regelen, waarop de ventilatorbladen kunnen worden aangesloten. Laten we beginnen met het aansluiten van het elektronische systeem volgens tabel 8.1.

DRV8835	Raspberry Pi Pico	Gelijkstroom motor
GND	GND	-
VIN, VCC en MD	VBUS	-
O1 en O2	-	twee benen
VM	-	-
IN1 PH	GP16	-
IN2 NL	GP17	-

Tabel 8.1: Voorbeeld van aansluiting van DRV8835 driver op Raspberry Pi Pico.

Open nu de Thonny editor en volg deze stappen:

1. Laten we beginnen met het kopiëren van het programma voor de DS18B20 temperatuursensor uit hoofdstuk 7.6.

```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime

one_wire_bus= onewire. OneWire( machine. Pin (13))
ds18b 20_sensor= ds18x20 . DS18X 20 ( one_wire_bus)
device_addr= ds18 b 20 _sensor. scan ()
```

```
9
10 terwijl True:
11     ds18b    20_sensor.      convert_temp      () utime .
12     sleep              (0,75) temp= ds18 b 20 _sensor.
13     read_temp ( device_addr [0]) print("T="+ str( temp ))
14
```

2. De snelheid van de gelijkstroommotor wordt geregeld door Pulsbreedtemodulatie (PWM) bij gebruik van de DRV8835 regelaar. Om dit te doen, configureren we pin 17 als PWM en pin 16 als GPIO-uitgang omdat deze wordt gebruikt om de draairichting te veranderen:

```
1  importeer
2  machine
3  importeer
4  onewire
5  importeer
6  ds18x20
7  importeer utime
8
9  one_wire_bus= onewire. OneWire( machine. Pin (13)) ds18b
10 20_sensor= ds18x20 . DS18X 20 ( one_wire_bus)
11 device_addr= ds18 b 20 _sensor. scan ()
12
13 DCmotor  =  machine . PWM ( machine . Pin (17))
14 richting= machine. Pin (16 , machine. Pin. OUT )
15
16 terwijl True:
17     ds18b    20_sensor.      convert_temp      () utime .
18     sleep              (0,75) temp= ds18 b 20 _sensor.
19     read_temp ( device_addr [0]) print("T="+ str( temp ))
```

3. Laten we nu de frequentie van het PWM-signaal instellen op 1 kHz en de draairichting (waarde 0 of 1):

```
1  importeer
2  machine
3  importeer
4  onewire
5  importeer
6  ds18x20
7  importeer utime
8
9  one_wire_bus= onewire. OneWire( machine. Pin (13)) ds18b
10 20_sensor= ds18x20 . DS18X 20 ( one_wire_bus)
11 device_addr= ds18 b 20 _sensor. scan ()
12
13 DCmotor  =  machine . PWM ( machine . Pin (17))
14 richting= machine. Pin (16 , machine. Pin. OUT )
15
16 DCmotor. freq (1000)
17 richting . waarde (0)
18
19 terwijl True:
20     ds18b    20_sensor.      convert_temp      () utime .
21     sleep              (0,75) temp= ds18 b 20 _sensor.
22     read_temp ( device_addr [0]) print("T="+ str( temp ))
```

4. Laten we nu variabelen maken waarin we de duty cycle van het PWM-signaal voor de minimum- en maximumsnelheid (RPM - omwentelingen per minuut) opslaan. Onthoud dat de functie *duty_u16()* waarden accepteert van 0 tot 65535, dus de variabelen moeten in dit bereik liggen. Daarnaast maken we variabelen om de minimum- en maximumtemperatuur op te slaan waarvoor de ventilatie moet werken. Laten we ook een mapping-functie maken waarmee we de temperatuur kunnen converteren naar de juiste PWM-belastingscyclus (enigszins evenredig met RPM):

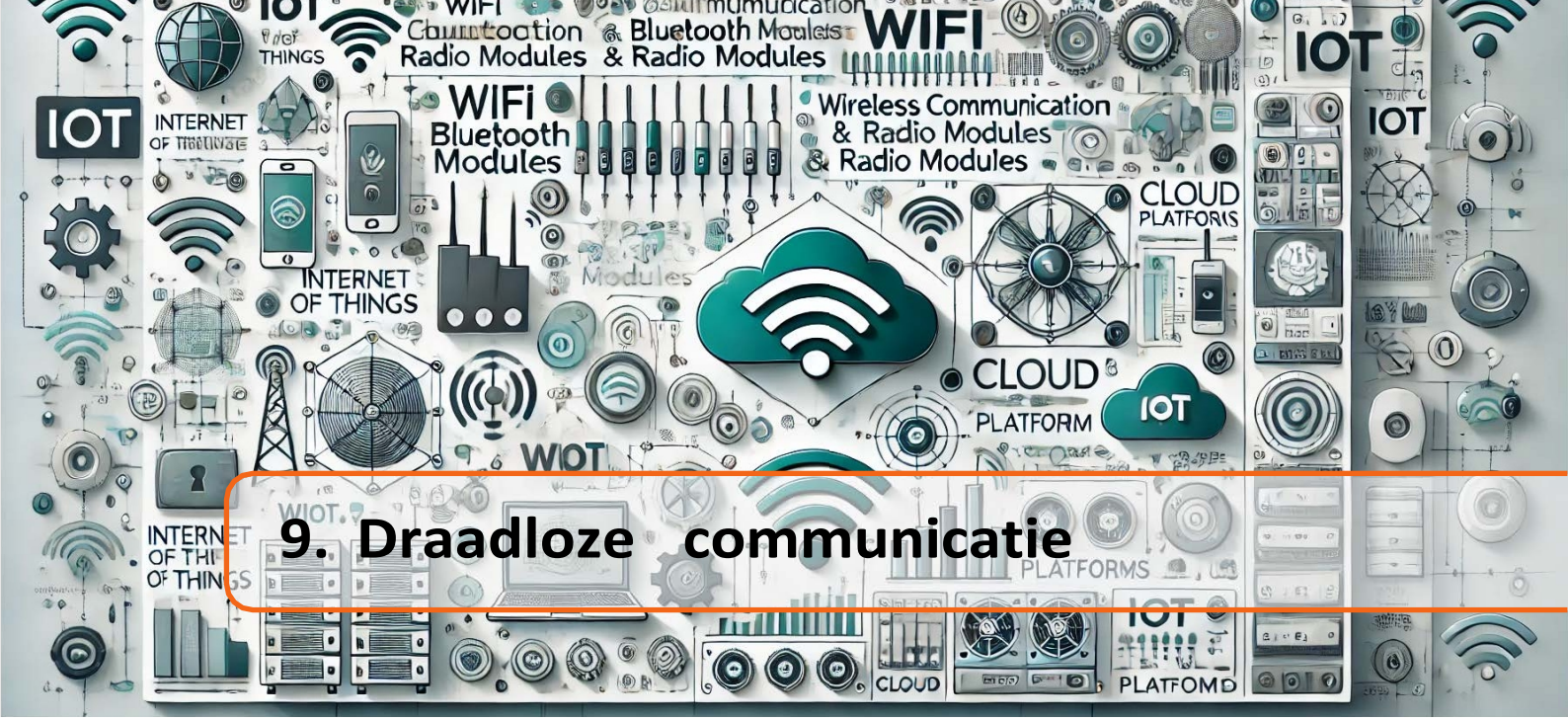
```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime

one_wire_bus= onewire. OneWire( machine. Pin (13)) ds18b
20_sensor= ds18x20 . DS18X 20 ( one_wire_bus)
device_addr= ds18 b 20 _sensor. scan ()

DCmotor  =  machine . PWM ( machine . Pin (17))
richting= machine. Pin (16 , machine. Pin. OUT )

DCmotor. freq (1000)
richting . waarde (0)

min_snelheid  =  20000
max_snelheid= 65535
Tmin =22
Tmax =50

def map_value (x, in_min , in_max , out_min , out_max ): return
    int(( x - in_min ) * ( out_max - out_min) / (
        →in_max - in_min )+ out_min )


terwijl True:
    ds18b    20_sensor.      convert_temp      () utime .
    sleep              (0,75) temp= ds18 b 20 _sensor.
    read_temp ( device_addr [0]) print("T="+ str( temp ))
```

5. De laatste stap is het omzetten van de temperatuur die wordt afgelezen van de DS18B20-sensor in de duty cycle van het PWM-signaal dat het motortoerental regelt, zodat hoe hoger de temperatuur, hoe sneller de motor de ventilatorbladen laat draaien. Vervolgens stellen we de aangewezen duty cycle in op PWM en wordt het juiste PWM-signaal gegenereerd op de PWM-pin:

```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime

one_wire_bus= onewire. OneWire( machine. Pin (13))
ds18b 20_sensor= ds18x20 . DS18X 20 ( one_wire_bus)
```

```
device_addr= ds18 b 20 _sensor. scan ()

DCmotor  =  machine . PWM ( machine . Pin (17))
richting= machine. Pin (16 , machine. Pin. OUT )

DCmotor. freq (1000) richting
. waarde (0)

min_snelheid  =  20000
max_snelheid= 65535
Tmin =22
Tmax =50

def map_value (x, in_min , in_max , out_min , out_max ): return
    int(( x - in_min ) * ( out_max - out_min) / (
        →in_max - in_min )+ out_min )

terwijl True:
    ds18b  20_sensor.  convert_temp  () utime .
    sleep  (0,75) temp= ds18 b 20 _sensor.
    read_temp ( device_addr [0]) print("T="+ str( temp ))

    snelheid= map_value( temp , Tmin , Tmax , min_snelheid ,
        '→  max_speed )
    DCmotor. duty_u16 (
    snelheid )

```

Het programma is nu klaar en je kunt het testen.



9. Draadloze communicatie

Draadloze communicatie kan op verschillende manieren via radiotechnieken, de populairste zijn: via Bluetooth of via Wi-Fi. In dit hoofdstuk bespreken we ze allebei. We zullen ook gebruik maken van een internetverbinding met de cloudservice - via de Adafruit IO cloud.

9.1 Voorbeeld 15: Bluetooth module

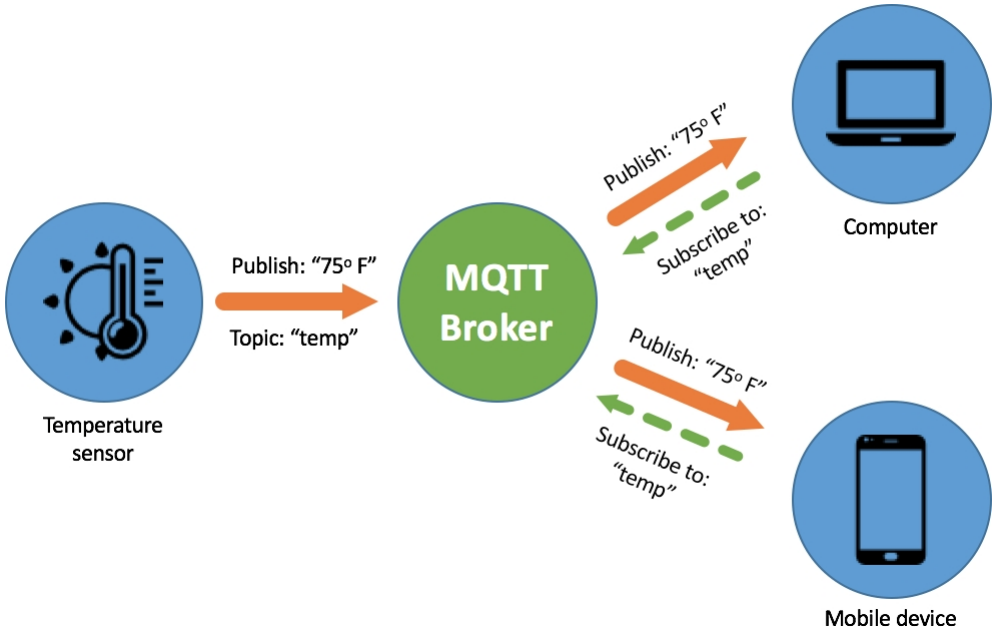
In het geval van het Raspberry Pi Pico W bord hebben we een ingebouwde Bluetooth module, die zeer goed gedocumenteerd is in de officiële datasheet: https://datasheets.raspberrypi.c om/picow/connecting-to-the-internet-with-pico-w.pdf . Ga naar de sectie "Werken met Bluetooth in MicroPython" en daar vind je een voorbeeld "Een temperatuurservice-randapparaat maken". Je kunt de voltooide code downloaden uit de repository: https://github.com/raspberrypi/pico-micropython-examples/tree/mast er/bluetooth(picow_ble_temp_sensor.py) en de benodigde *ble_advertising.py* bibliotheek. De bibliotheek moet worden geladen op het Raspberry Pi Pico-bord en voer gewoon het *picow_ble_temp_sensor*.py-voorbeeld uit. Om de gegevens uit te lezen kun je de LightBlue-applicatie gebruiken op een smartphone met Android of iOS.

9.2 Voorbeeld 16: Adafruit IO

In dit voorbeeld sturen we de temperatuurmeting van de DS18B20 sensor naar de Adafruit IO cloud. Er zijn verschillende clouds, maar in deze handleiding zullen we ons concentreren op de cloud die volgens ons het eenvoudigst is en veel mogelijkheden biedt voor het maken van IoT-projecten. De Adafruit IO cloud gebruikt het MQTT (Message Queuing Telemetry Transport) proto- col, dat gebaseerd is op het publish/subscribe model. Het idee is dat client "A" informatie publiceert onder een bepaald topic en dat elke andere client die zich heeft geabonneerd op dit topic zal de gegevens lezen die door client "A" zijn gepubliceerd. Dit idee wordt schematisch weergegeven in Fig. 9.1.

Om een programma te maken dat temperatuur naar de Adafruit IO cloud stuurt, volg je deze stappen:

1. Eerst moet je een gratis account aanmaken op https://io.adafruit.com.
2. Vervolgens moet je de temperatuurwaarde naar de cloud sturen. Om dit te doen, moet een feed maken. Een feed is een object dat gegevens opslaat. Om een feed te maken, ga je naar het tabblad "Feed" en selecteer je de knop "Nieuwe feed" (zie Afb. 9.2).



Afbeelding 9.1: Hoe het MQTT-protocol werkt. Bron: https://www.akcp.com/blog/sc aling-mqtt-netwerk-voor-betere-werking/

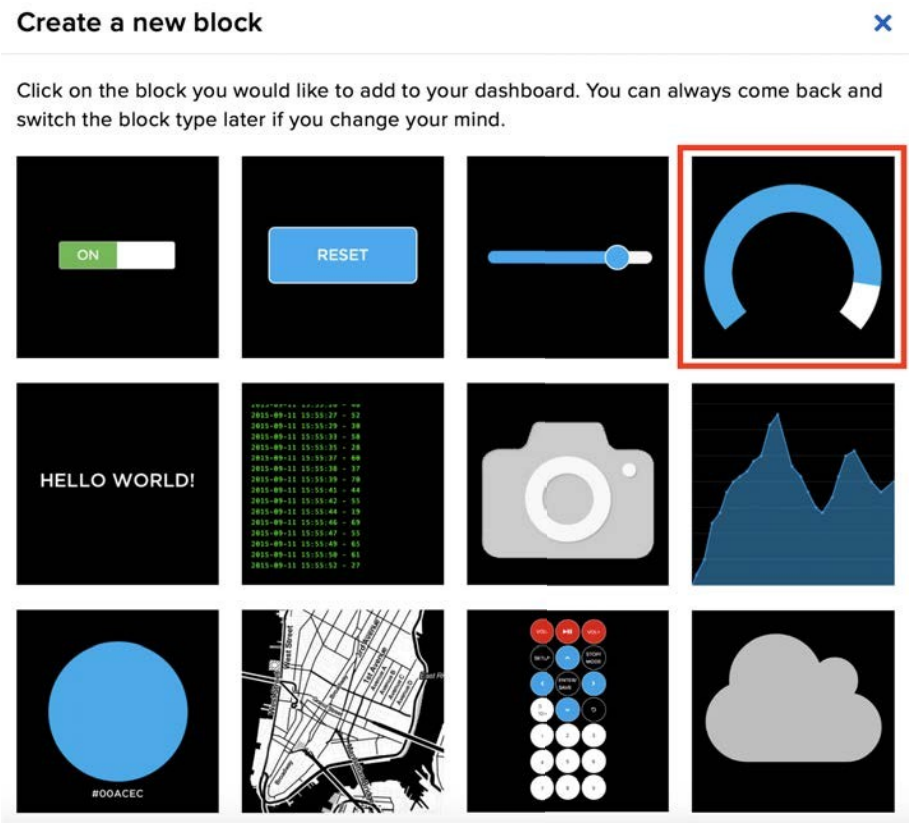
Afbeelding 9.2: Een feed aanmaken in Adafruit IO.

3. Dan verschijnt er een venster waarin je de naam van de feed moet invoeren, bijvoorbeeld *Temp*.
4. De volgende stap is het maken van een dashboard. Selecteer hiervoor de tab "Dashboards" en vervolgens "New Dashboard" (zie Afb. 9.3).

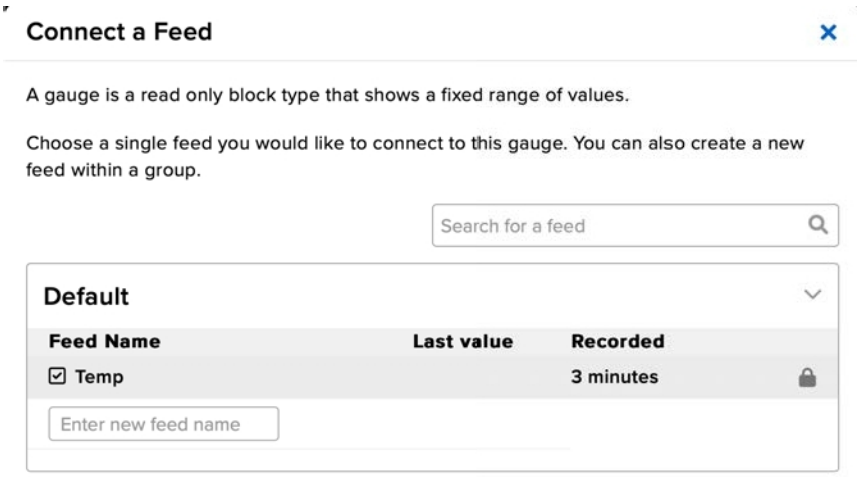
Figuur 9.3: Een dashboard aanmaken in Adafruit IO.

5. Er verschijnt een venster waarin u de geselecteerde dashboardnaam moet invoeren. Selecteer vervolgens aan de rechterkant het symbool Instellingen en kies "Nieuw blok maken" (zie Afb. 9.4).
6. Adafruit IO heeft verschillende blokken beschikbaar om gegevens weer te geven (tekstvakken, meters, grafieken, kaarten, enz.). In ons geval kiezen we voor een meter (zie Fig. 9.5).

Afbeelding 9.4: Een dashboard bewerken in Adafruit IO.



Afbeelding 9.5: Beschikbare blokken in Adafruit IO.



Afbeelding 9.6: Het blok aansluiten op de voeding. in Adafruit IO.

7. Vervolgens verschijnt er een venster met de vraag op welke voeding we het blok willen aansluiten. Selecteer de feed die je hebt gemaakt om de temperatuurwaarden op te slaan (zie Fig. 9.6).
8. Laten we nu naar de Thonny editor gaan en de *umqtt.simple* bibliotheek installeren vanuit de packages manager in de Thonny editor.
9. Laten we nu terugkeren naar het voorbeeld van het uitlezen van de temperatuur van de DS18B20-sensor:

```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime

one_wire_bus= onewire. OneWire( machine. Pin (13)) ds18b
20_sensor= ds18x20 . DS18X 20 ( one_wire_bus)
device_addr= ds18 b 20 _sensor. scan ()

terwijl True:
    ds18b    20_sensor.    convert_temp    () utime .
    sleep            (0,75) temp= ds18 b 20 _sensor.
    read_temp (device_addr [0]) print("T="+ str( temp ))
```

10. Voeg vervolgens de benodigde bibliotheken toe:

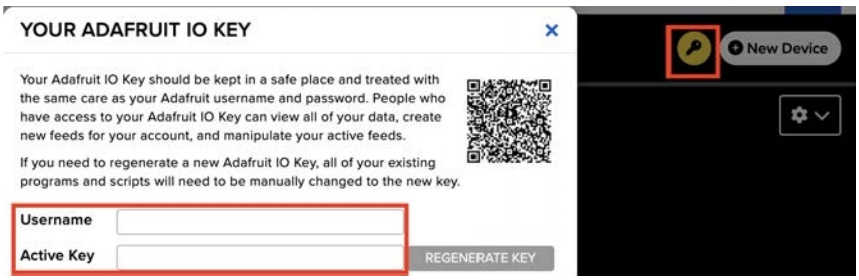
```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime

van umqtt.simple importeer

one_wire_bus= onewire. OneWire( machine. Pin (13)) ds18b
20_sensor= ds18x20 . DS18X 20 ( one_wire_bus)
device_addr= ds18 b 20 _sensor. scan ()

terwijl True:
    ds18b    20_sensor.    convert_temp    () utime .
    sleep            (0,75) temp= ds18 b 20 _sensor.
    read_temp (device_addr [0]) print("T="+ str( temp ))
```

11. Voeg een stukje code toe waarmee je verbinding kunt maken met je Wi-Fi. Hier moet je gevoelige gegevens invullen in regel 12-13. Voer eerst de naam van je Wi-Fi (SSID - Service Set Identifier) en vervolgens het wachtwoord in, zodat de Raspberry Pi Pico W verbinding kan maken met je Wi-Fi-netwerk. Deze Wi-Fi moet verbinding met het internet mogelijk maken, omdat we verbinding willen maken met de Adafruit IO Cloud-service:

```
importeer machine
importeer onewire
importeer ds18x20
importeer utime
importeer netwerk
van umqtt. simple importeer MQTTClient
```

```
7
8  one_wire_bus= onewire. OneWire( machine. Pin (13))
9  ds18b 20_sensor= ds18x20 . DS18X 20 ( one_wire_bus)
10 device_addr= ds18 b 20 _sensor. scan ()
11
12 WIFI_SSID  =   " uw_wifi_naam "
13 WIFI_PASSWORD=" your_wifi_password " ADAFRUIT_IO_USERNAME="
   gebruikersnaam "
14 ADAFRUIT_IO_KEY=" sleutel "
15
16 def connect_wifi ():
17       wlan= network . WLAN ( network. STA_IF )
18       wlan.              active(              True) wlan .
19       connect( WIFI_SSID , WIFI_PASSWORD ) while
20       not wlan . isconnected ():
21            print(" Verbinding maken met Wi -
22            Fi...") utime. sleep (1)
23        print(" Verbonden met Wi - Fi:", wlan . ifconfig ())
24
25 connect_wifi ()
26
27 terwijl True:
28       ds18b    20_sensor.      convert_temp      () utime .
29       sleep              (0,75) temp= ds18 b 20 _sensor.
30       read_temp ( device_addr [0]) print("T="+ str( temp ))
31
32
```

12. De laatste twee parameters zijn referenties voor toegang tot de Adafruit IO cloud. Ga terug naar de Adafruit website en klik op het sleutelsymbool. Wanneer je dit doet, verschijnen je Gebruikersnaam en Sleutel (zie Fig. 9.7). Kopieer deze gegevens naar het programma op regel 14-15.



Afbeelding 9.7: Gebruikersnaam en sleutel.

13. Nu zullen we het MQTT protocol gebruiken. Gebruik de onderstaande code en verander alleen de naam van de feed in regel 30:

```
1 importeer machine
2 importeer onewire
3 importeer ds18x20
4 importeer utime
5 importeer netwerk
6 van umqtt. simple importeer MQTTClient
7
```

```
one_wire_bus= onewire. OneWire( machine. Pin (13))
ds18b 20_sensor  =  ds18x20 . DS18X 20 ( een_draadse_bus)
device_addr= ds18 b 20 _sensor. scan ()

WIFI_SSID=" uw_wifi_naam "
WIFI_PASSWORD=" uw_wifi_wachtwoord "
ADAFRUIT_IO_USERNAME=" gebruikersnaam "
ADAFRUIT_IO_KEY=" sleutel "

def connect_wifi ():
      wlan  =  network . WLAN ( network. STA_IF )
      wlan. actief( True)
      wlan . connect( WIFI_SSID , WIFI_PASSWORD )
      terwijl niet wlan . isconnected ():
            print(" Verbinding maken met Wi - Fi...")
            utime. slaap (1)
      print(" Verbonden met Wi - Fi:", wlan . ifconfig ())

ADAFRUIT_IO_SERVER=". adafruit. com "
ADAFRUIT_IO_PORT  =  1883       #  poort MQTT
CLIENT_ID  =  " framboos_pi_pico"

FEED_TEMP_LEVEL=" gebruikersnaam/ feeds/ Temp "

client= MQTTClient( CLIENT_ID , ADAFRUIT_IO_SERVER ,
    →ADAFRUIT_IO_PORT ,  ADAFRUIT_IO_USERNAME ,
    →ADAFRUIT_IO_KEY )

def  connect_mqtt ():
      client. verbinden ()
      print("  verbonden met  Adafruit  IO")

connect_wifi ()
verbind_mqtt ()

terwijl  True:
      ds18b 20_sensor. convert_temp ()
      utime . slaap (0,75)
      temp= ds18 b 20 _sensor. read_temp ( device_addr [0])
      print("T="+ str( temp ))
```

14. Laten we nu een functie toevoegen om gegevens naar de cloud te sturen:

```
importeer machine
importeer onewire
importeer ds18x20
importeer utime
importeer netwerk
van umqtt. simple importeer MQTTClient

one_wire_bus= onewire. OneWire( machine. Pin (13))
```

```
ds18b20_sensor = ds18x20.DS18X20(een_draadse_bus)
device_addr= ds18b20_sensor.scan()

WIFI_SSID="uw_wifi_naam"
WIFI_PASSWORD="uw_wifi_wachtwoord"
ADAFRUIT_IO_USERNAME="gebruikersnaam"
ADAFRUIT_IO_KEY="sleutel"

def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.actief(True)
    wlan.connect(WIFI_SSID,WIFI_PASSWORD)
    terwijl niet wlan.isconnected():
        print("Verbinding maken met Wi-Fi...")
        utime.slaap(1)
    print("Verbonden met Wi-Fi:", wlan.ifconfig())

ADAFRUIT_IO_SERVER=".adafruit.com"
ADAFRUIT_IO_PORT = 1883       # poort MQTT
CLIENT_ID = "framboos_pi_pico"

FEED_TEMP_LEVEL="gebruikersnaam/feeds/Temp"

client= MQTTClient(CLIENT_ID,ADAFRUIT_IO_SERVER,
    →ADAFRUIT_IO_PORT, ADAFRUIT_IO_USERNAME,
    →ADAFRUIT_IO_KEY)

def connect_mqtt():
    client.verbinden()
    print("verbonden met Adafruit IO")

connect_wifi()
verbind_mqtt()

def send_data(temp):
    client.publiceer(FEED_TEMP_LEVEL,str(temp))
    print("Temp. verzonden:"+str(temp))

terwijl True:
    ds18b20_sensor.convert_temp()
    utime.sleep(0,75)
    temp= ds18b20_sensor.read_temp(device_addr[0])
    print("T="+str(temp))
```

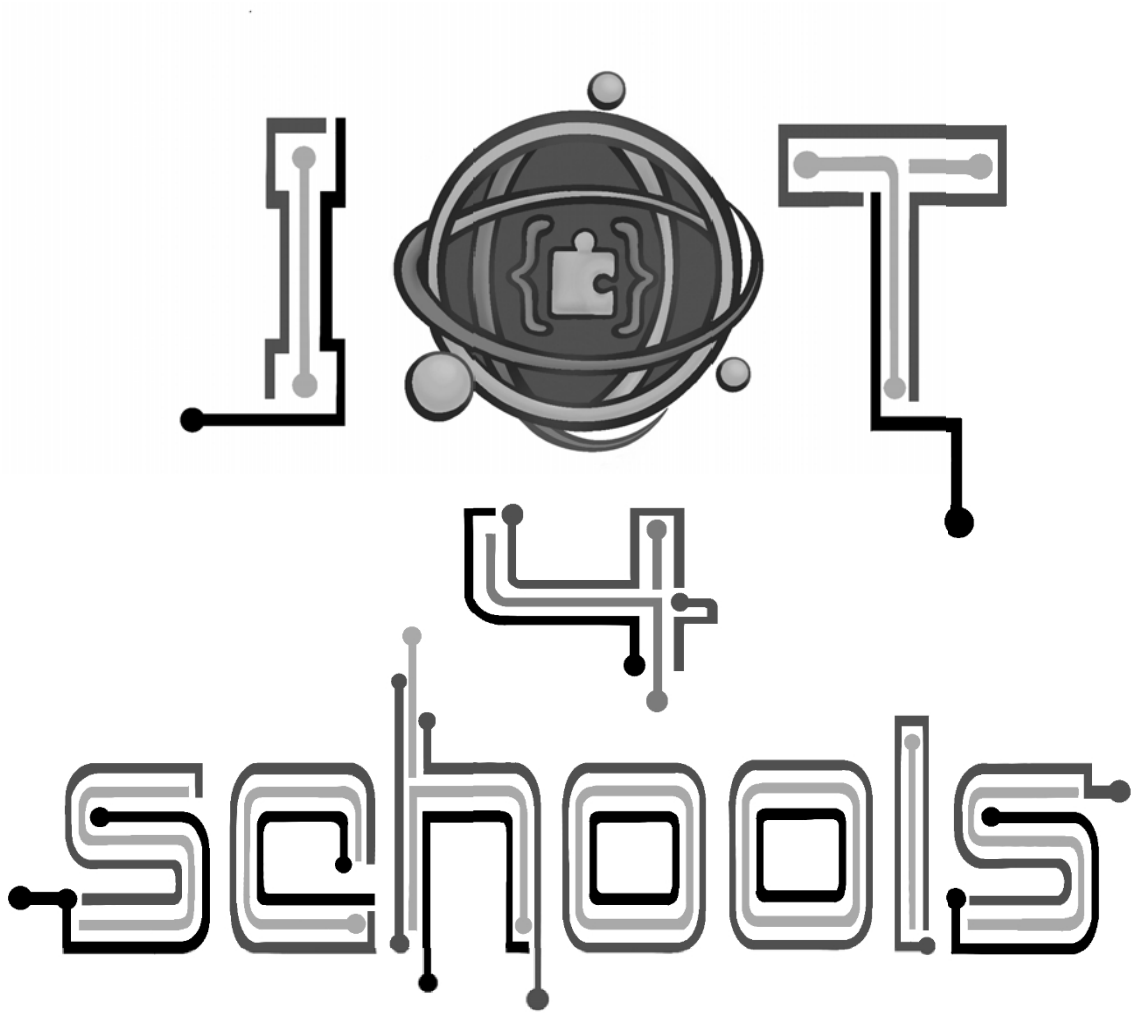
15. De laatste stap is het aanroepen van de functie voor het verzenden van gegevens na het meten van de temperatuur en het verzenden van de waarde naar de cloud:

```
importeer
machine
importeer
onewire
importeer
ds18x20
importeer utime
```

```
  netwerk importeren
van umqtt.simple importeer MQTTClient

one_wire_bus=onewire.OneWire(machine.Pin(13))
ds18b20_sensor = ds18x20.DS18X20(een_draadse_bus)
device_addr=ds18b20_sensor.scan()

WIFI_SSID="uw_wifi_naam"
WIFI_PASSWORD="uw_wifi_wachtwoord"
ADAFRUIT_IO_USERNAME="gebruikersnaam"
ADAFRUIT_IO_KEY="sleutel"

def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.actief(True)
    wlan.connect(WIFI_SSID,WIFI_PASSWORD)
    terwijl niet wlan.isconnected():
        print("Verbinding maken met Wi-Fi...")
        utime.slaap(1)
    print("Verbonden met Wi-Fi:", wlan.ifconfig())

ADAFRUIT_IO_SERVER=".adafruit.com"
ADAFRUIT_IO_PORT = 1883       # poort MQTT
CLIENT_ID = "framboos_pi_pico"
FEED_TEMP_LEVEL="gebruikersnaam/feeds/Temp"

client=MQTTClient(CLIENT_ID,ADAFRUIT_IO_SERVER,
    →ADAFRUIT_IO_PORT, ADAFRUIT_IO_USERNAME,
    →ADAFRUIT_IO_KEY)

def connect_mqtt():
    client.verbinden()
    print("verbonden met Adafruit IO")

connect_wifi()
verbind_mqtt()

def send_data(temp):
    client.publiceren(FEED_TEMP_LEVEL,str(temp))
    print("Temp. verzonden:"+str(temp))

terwijl True:
    ds18b20_sensor.convert_temp()
    utime.slaap(0,75)
    temp=ds18b20_sensor.read_temp(device_addr[0])
    print("T="+str(temp))
    verzend_gegevens(temp)
```

Het programma is klaar. Voer het uit en ga naar Adafruit IO naar het aangemaakte dashboard. Je zou de gemeten temperatuur op de indicator moeten zien.

Dit was het laatste voorbeeld in deze handleiding. We hopen dat de gepresenteerde voorbeelden je in staat hebben gesteld om het Raspberry Pi Pico W board te leren kennen. We moedigen je aan om door te gaan met ontwikkelen en te zoeken naar interessante voorbeelden van het gebruik van het Raspberry Pi Pico W board, zowel binnen het IoT4schools project (https://www.iot.fizyka.pw.edu.pl) als elders.


IOT
4
schools