

# MicroPython op de ESP8266

Dit document en de bijhorende source code kan je [hier](#) downloaden. Enjoy !

## Inhoudsopgave

MicroPython.....	5
Python voor Microcontrollers.....	5
Kenmerken.....	5
MicroPython versus Arduino.....	5
Ondersteunde hardware.....	6
ESP8266.....	8
Kenmerken.....	8
Firmware.....	8
Modules.....	9
AI-Thinker.....	9
Andere.....	10
Projecten.....	11
NodeMCU ESP8266.....	11
Kenmerken.....	11
Aansluitingen.....	12
Wemos D1 Mini.....	13
Kenmerken.....	13
Aansluitingen.....	13
Project 1 : MicroPython flashen op de ESP8266.....	15
Installatie flashtool.....	15
Installatie firmware.....	15
Project 2 : Interne LED laten knipperen.....	17
Opgave.....	17

Materiaal.....	17
Schakeling.....	17
Code.....	18
Project 3: Externe LED laten knipperen.....	19
Opgave.....	19
Materiaal.....	19
Schakeling.....	19
Code.....	20
Project 4 : Faden van een externe LED met PWM.....	21
Opgave.....	21
Materiaal.....	21
Schakeling.....	21
Pulse Width Modulation.....	22
Code.....	23
Project 5 : Schakelen van een LED met een drukknop.....	24
Opgave.....	24
Materiaal.....	24
Schakeling.....	24
Pull up – pull down weerstand.....	25
Code.....	25
Project 6 : Sturen van een servomotor.....	26
Opgave.....	26
Materiaal.....	26
Schakeling.....	26
Servomotor.....	27
Code.....	27
Project 7 : Sturen van een stappemotor met de ULN2003A.....	28
Opgave.....	28
Materiaal.....	28
Schakeling.....	28
Stappemotor.....	29
Code.....	29
Project 8 : Sturen van een DC motor met een H-bridge.....	32
Opgave.....	32
Materiaal.....	32
Schakeling.....	32
H-bridge.....	33
Code.....	33
Project 9 : Een melodietje afspelen op een speaker.....	37
Opgave.....	37
Materiaal.....	37
Schakeling.....	37
Piëzo.....	38
Code.....	38
Project 10 : Uitlezen van een DH11/DT22 sensor.....	39
Opgave.....	39
Materiaal.....	39
Schakeling.....	39
Code.....	40
Project 11 : Uitlezen analoge poort van de ESP8266.....	41

Opgave.....	41
Materiaal.....	41
Schakeling.....	41
Spanningsdeler.....	42
Code.....	42
Project 12 : Analoge waarde van een ADC uitlezen via SPI.....	43
Opgave.....	43
Materiaal.....	43
Schakeling.....	43
Serial Peripheral Interface (SPI).....	44
Code.....	45
Project 13 : LED matrix aansturen via SPI.....	47
Opgave.....	47
Materiaal.....	47
Schakeling.....	47
Code.....	48
Project 14 : OLED display aansturen met I2C.....	51
Opgave.....	51
Materiaal.....	51
Schakeling.....	51
I2C.....	52
Code.....	53
Project 15 : LCD karakter display parallel aansturen.....	54
Opgave.....	54
Materiaal.....	54
Schakeling.....	54
Dot matrix LCD.....	55
Code.....	55
Project 16 : LCD karakter display aansturen met I2C.....	59
Opgave.....	59
Materiaal.....	59
Schakeling.....	59
I2C backpack module.....	60
Code.....	60
Project 17 : NeoPixel matrix aansturen.....	67
Opgave.....	67
Materiaal.....	67
Schakeling.....	67
NeoPixel.....	67
Code.....	68
Project 18 : ESP8266 als WiFi client.....	69
Opgave.....	69
Materiaal.....	69
Code.....	69
Project 19 : ESP8266 als Socket Server.....	73
Opgave.....	73
Materiaal.....	73
Schakeling.....	73
Code.....	74
Project 20 : ESP8266 als Socket Client.....	79

Opgave.....	79
Materiaal.....	79
Schakeling.....	79
Code.....	80
Project 21 : ESP8266 als remote mp3 speler.....	83
Opgave.....	83
Materiaal.....	83
Schakeling.....	83
DFR0299.....	83
Code.....	84
Datasheets.....	94
ESP8266.....	94
Tower Pro MG90S.....	94
28BYJ-48.....	94
ULN200xA.....	94
L298.....	94
DHT11 / DHT22.....	94
LDR.....	94
MCP3004/3008.....	94
MAX7219.....	94
SSD1306.....	94
HD44780U.....	94
PCF8574.....	94
WS2812B.....	95
DFR0299.....	95
Interessante websites.....	96

# MicroPython

## Python voor Microcontrollers

MicroPython is een op Python 3 gebaseerde programmeertaal die speciaal ontworpen is voor microcontroller ontwikkelingsbordjes. Deze bordjes hebben beperkte resources op gebied van geheugen en processorkracht.

Je kan met MicroPython simpele en verstaanbare code schrijven, in tegenstelling tot de Arduino die gebruik maakt van complexe talen als C en C++. Dit maakt het dan ook beter geschikt voor beginnende programmeurs. Bemerk echter dat MicroPython bijna net zo uitgebreid is als zijn grote broer Python en dat dus ook gevorderde Python programmeurs deze versie zullen smaken.

## Kenmerken

MicroPython heeft een aantal unieke kenmerken die andere ingebetde systemen niet hebben :

- *Interactieve read-evaluate-print loop (REPL).*

Deze is vergelijkbaar met de IDLE omgeving van zijn grote broer Python. Het laat je toe interactief commando's uit te voeren zonder te moeten compileren of uploaden naar het bordje. Ideaal om te experimenteren !

- *Uitgebreide software bibliotheek.*

Zoals zijn grote broer zijn er heel wat bibliotheken beschikbaar voor allerlei soorten taken. Parsing JSON data van web services, tekst zoeken met reguliere expressies, netwerk socket programmeren, GPIO, PWM, ADC, I2C, SPI, WiFi support, Neopixels, LED strips, OLED displays, ... het is allemaal beschikbaar met de ingebouwde bibliotheken.

- *Uitbreidbaarheid.*

MicroPython is een interpreter en de code moet eerst omgezet worden naar iets dat de microcontroller kan uitvoeren. Daarom is het voor tijdkritische zaken minder geschikt. Het kan echter uitgebreid worden met C/C++ functies om zo de uitvoersnelheid gevoelig te verhogen.

## MicroPython versus Arduino

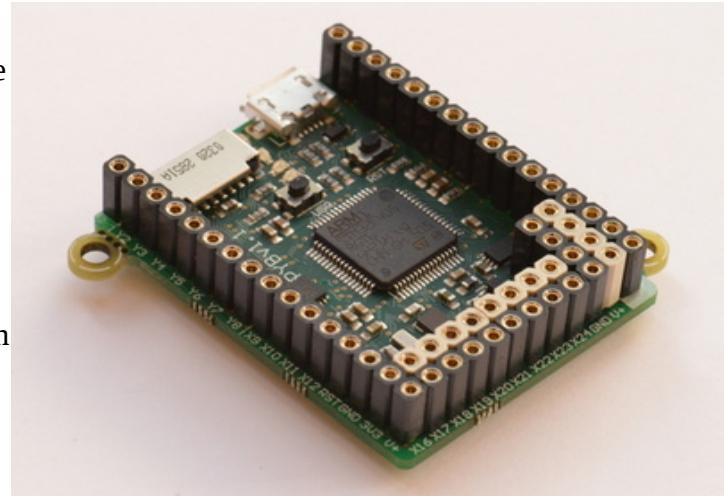
MicroPython	Arduino
Is enkel een programmeertaal interpreter. Het bevat geen ontwikkelomgeving. Sommige ondersteunde hardware hebben wel een webgebaseerde editor.	Is een volledige omgeving met de IDE ontwikkelomgeving, de programmeertaal gebaseerd op C/C++ en de hardware (vb Arduino UNO R3)
De code wordt interactief door de interpreter omgezet in uitvoerbare code. Hierdoor is de code wat minder performant qua snelheid en geheugengebruik.	De code wordt gecompileerd en upgeload naar de hardware. Zodoende wordt deze uitgevoerd aan maximale snelheid.

# Ondersteunde hardware

Er zijn reeds vele ontwikkelingsbordjes die ondersteund worden door MicroPython. Voor de laatste stand van zaken raadpleeg je best de [MicroPython website](#).

- **Pyboard**

Dit is het officiële MicroPython bordje gemaakt door de ontwerpers van MicroPython. Het biedt dan ook volledige ondersteuning voor de aanwezige hardware. Het is gebaseerd op een Cortex M4 CPU met 1 MB ROM en 192 KB RAM en heeft oa een micro USB aansluiting, een microSD kaart slot, RTC, GPIO , ADC, DAC, Leds, switches en een ingebouwde 3.3V spanningsregelaar.



- **ESP8266**

Voor deze populaire ESP8266 WiFi microcontroller is de MicroPython ondersteuning excellent. GPIO, ADC, PWM, I<sup>2</sup>C en SPI zijn beschikbaar. Daarenboven is WiFi en internet toegang goed ondersteund. Er is zelfs een web gebaseerde REPL beschikbaar zodat je via je browser het bordje kan programmeren. Er zijn zeer veel bordjes met deze microcontroller op de markt te koop. Hiernaast de NodeMCU variant van DoIt.



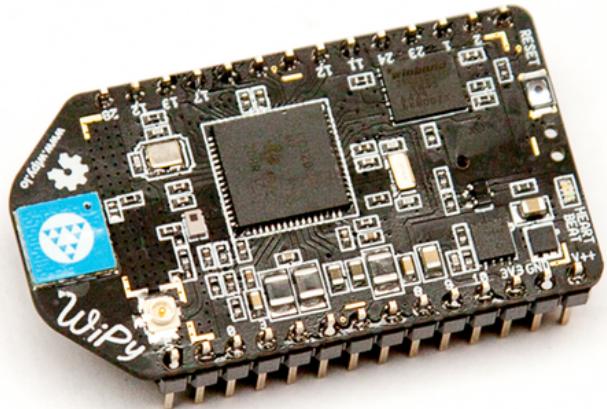
- **SAMD21**

De Atmel SAMD21 gebaseerde bordjes zoals de Adafruit Feather M0 en de Arduino Zero worden recent ondersteund door een experimentele MicroPython versie die Adafruit aan het ontwikkelen is. Meer informatie in de [MicroPython SAMD21 guide](#).



- **WiPy**

De WiPy is ontwikkeld door [Pycom](#), een Nederlands bedrijf en heeft net zoals de ESP8266 uitstekende WiFi ondersteuning. Het belangrijkste kenmerk is echter dat er een mooie ontwikkelomgeving wordt aangeboden om te coderen en de code op het bordje te krijgen.



- **BBC micro:bit**

Deze [micro:bit](#) microcontroller is net zoals de Raspberry Pi bedoeld om de Britse schoolgaande jeugd kennis te laten maken met Internet of Things (IoT) en hoe deze te programmeren. Met dit doel zijn er ondertussen reeds 1 miljoen exemplaren gratis verdeeld aan 11- en 12-jarige studenten in de Britse scholen. Er zijn [verschillende ontwikkelomgevingen](#) beschikbaar op hun website die het programmeren en uploaden van de code makkelijk maken.



# ESP8266

De ESP8266 is een goedkope WiFi chip met een complete TCP/IP stack en een ingebouwde microcontroller, gemaakt door Espressif Systems, een Chinese producent in Shanghai.

Deze kleine module kan verbinding maken met een draadloos netwerk met behulp van Hayes-achtige commando's (AT commando's zoals de modems vroeger gebruikten).

Aanvankelijk was er enkel Chinese documentatie beschikbaar. Daar kwam in 2014 verandering in toen Westerse hobbyisten de module ontdekten en ermee aan de slag gingen in hun projecten. De Chinese documentatie werd vertaald en mede door de lage kostprijs werd de ESP8266 een grote hit in tal van IoT (Internet of Things) projecten.

## Kenmerken

- 32-bit microcontroller Tensilica Xtensa LX106 op 80 MHz
- 64 KB instructie RAM, 96 KB data RAM
- 512 KB tot 4 MB flash RAM
- IEEE 802.11 b/g/n WiFi met WEP en WPA/WPA2 authenticatie
- 16 GPIO pinnen
- SPI en I<sup>2</sup>C interface
- I<sup>2</sup>S interface voor verbinding met digitale audio
- UART
- 10bit analoog naar digitaal convertor

## Firmware

Naast de Software Development Kits (SDK) van de producent Espressif zijn er nog verschillende open source firmwares beschikbaar :

- *Arduino* : heeft een C++ gebaseerde firmware beschikbaar. Na het laden van de ESP8266 bibliotheek in de Arduino IDE kan je de module via deze IDE makkelijk programmeren.
- *NodeMCU* : met deze firmware kan je de ESP8266 programmeren in de Lua scripttaal.
- *MicroPython* : dit is een implementatie van Python 3 die speciaal gemaakt is voor microcontrollers met beperkte resources (geheugen en processorkracht). Er bestaat ook een goed ondersteunde versie voor de ESP8266.
- *ESP8266 BASIC* : is een basic interpreter speciaal gemaakt voor IoT (Internet of Things)
- *Mongoose* : deze heeft een bijhorende cloud service.

# Modules

## AI-Thinker

Dit zijn de meest verspreide modules. Om die te kunnen programmeren is er echter nog een externe 3.3V voeding nodig, alsook een seriële TTL naar USB adapter voor communicatie met de ontwikkelomgeving op de host computer. Eenmaal de ontwikkeling gebeurd is en de firmware geladen zijn deze modules ideaal om je project live te zetten.

Name	Active pins	Pitch	Form factor	LEDs	Antenna	Shielded ?	Dimensions (mm)	Notes
ESP-01	6	0.1"	2×4 DIL	Yes	PCB trace	No	14.3 × 24.8	
ESP-02	6	0.1"	2×4 castellated	No	U-FL connector	No	14.2 × 14.2	
ESP-03	10	2 mm	2×7 castellated	No	Ceramic	No	17.3 × 12.1	
ESP-04	10	2 mm	2×4 castellated	No	None	No	14.7 × 12.1	
ESP-05	3	0.1"	1×5 SIL	No	U-FL connector	No	14.2 × 14.2	
ESP-06	11	misc	4×3 dice	No	None	Yes	14.2 × 14.7	Not FCC approved
ESP-07	14	2 mm	2×8 pinhole	Yes	Ceramic + U-FL connector	Yes	20.0 × 16.0	Not FCC approved
ESP-08	10	2 mm	2×7 castellated	No	None	Yes	17.0 × 16.0	Not FCC approved
ESP-09	10	misc	4×3 dice	No	None	No	10.0 × 10.0	
ESP-10	3	2 mm?	1×5 castellated	No	None	No	14.2 × 10.0	
ESP-11	6	0.05"	1×8 pinhole	No	Ceramic	No	17.3 × 12.1	
ESP-12	14	2 mm	2×8 castellated	Yes	PCB trace	Yes	24.0 × 16.0	FCC and CE approved
ESP-12E	20	2 mm	2×8 castellated	Yes	PCB trace	Yes	24.0 × 16.0	4 MB Flash
ESP-12F	20	2 mm	2×8 castellated	Yes	PCB trace	Yes	24.0 × 16.0	FCC and CE approved. Improved antenna performance. 4 MB Flash
ESP-13	16	1.5 mm	2×9 castellated	No	PCB trace	Yes	W18.0 x L20.0	Marked as "FCC". Shielded module is placed sideways, as compared to the ESP-12 modules.
ESP-14	22	2 mm	2×8 castellated +6	No	PCB trace	Yes	24.3 x 16.2	

## Andere

Vele van deze modules hebben naast een ESP-xx module ook een UART naar USB adapter, een micro USB connector en een 3.3V regulator die de 5V van de USB connectie omzet naar de juiste spanning voor de ESP chip. Dit zijn ideale bordjes om je project te prototypen. Eenmaal je project af is, kies je best een goedkopere basis ESP module om het live te zetten.

Dit zijn er enkele van de vele :

Name	Pins	ESP-xx	Shields?
<a href="#">Geekcreit Doit NodeMcu Lua ESP8266</a>	14	ESP-12E	Nee
<a href="#">Adafruit ESP8266 Feather Huzzah</a>	14	ESP-12	Ja
<a href="#">WeMos D1 Mini</a>	12	ESP-12F	Ja

# Projecten

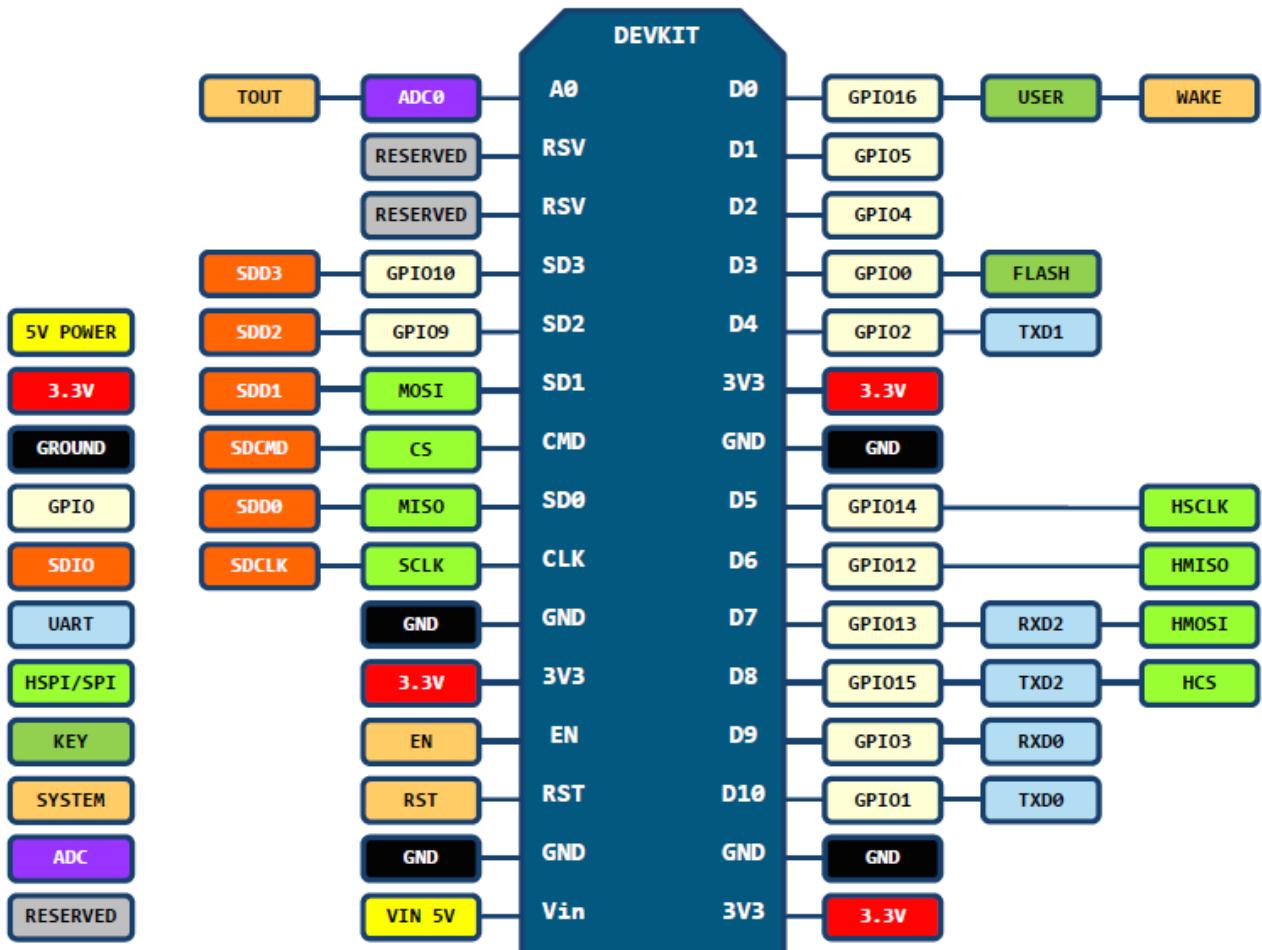
## NodeMCU ESP8266



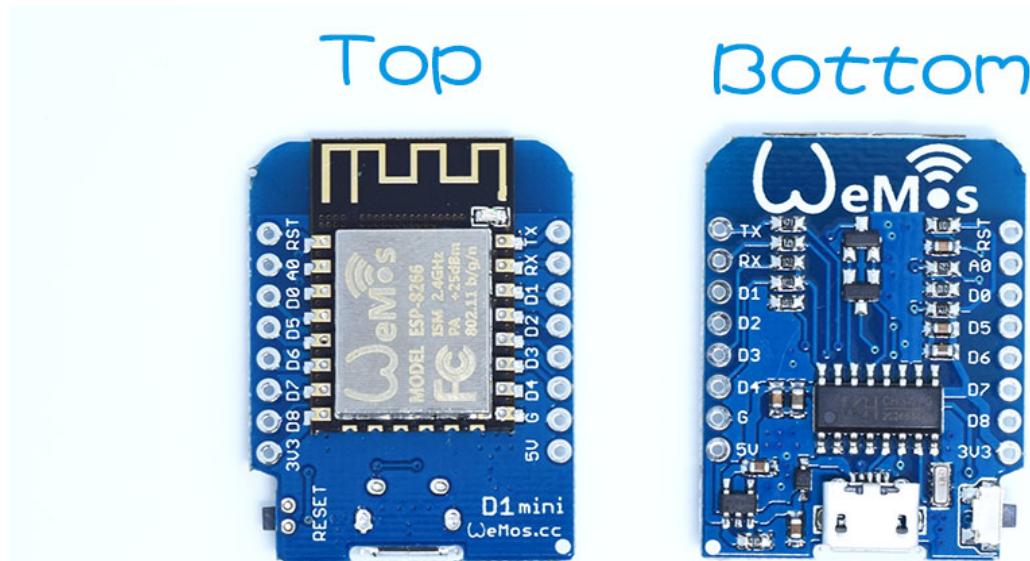
### Kenmerken

- Wireless 802.11 b / g / n standard
- Support STA / AP / STA + AP three operating modes
- Built-in TCP / IP protocol stack to support multiple TCP Client connections (5 max)
- D0 used as GPIO, no PWM !
- D1 ~ D8, SD1 ~ SD3: used as GPIO, PWM, I2C, etc., port driver capability 15mA
- ADC0: 1 channel ADC
- Power input: 4.5V ~ 9V (10V max), USB-powered
- Current: continuous transmission: ≈70mA (200mA max), Standby: <200uA
- Transfer rate: 110 – 460800 bps
- Support UART / GPIO data communication interface
- Remote firmware upgrade (OTA)
- Support Smart Link Smart Networking
- Working temperature: -40 °C ~ + 125 °C
- Drive Type: Dual high-power H-bridge driver

## Aansluitingen



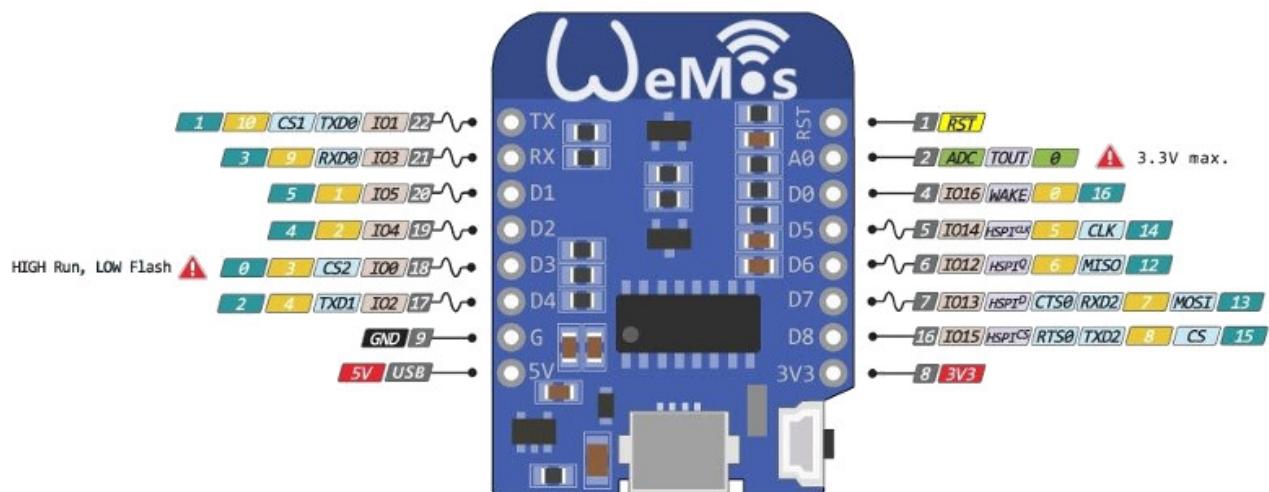
# Wemos D1 Mini



## Kenmerken

- Mini WiFi bordje gebaseerd op de ESP8266 12F (34,2 x 56,6 mm)
- Voeding 3,3 V via ingebouwde spanningsregelaar
- 11 digitale GPIO pinnen met interrupt/PWM/I<sup>2</sup>C/1wire support (uitgezonderd D0)
- 1 analoge GPIO pin (max 3.2 V input)
- Micro USB connector voor voeding en programmatie
- Compatibel met Arduino
- Compatibel met NodeMC
- Diverse shields kunnen op dit bordje ingeplugged worden

## Aansluitingen



<b>D1 Mini Pin</b>	<b>Function</b>	<b>ESP8266 Pin</b>
TX	TXD	TXD
RX	RXD	RXD
A0	Analog input, max 3.3V input	A0
D0	IO	GPIO16
D1	IO, SCL	GPIO5
D2	IO, SDA	GPIO4
D3	IO, 10k Pull-up	GPIO0
D4	IO, 10k Pull-up, BUILTIN_LED	GPIO2
D5	IO, SCK	GPIO14
D6	IO, MISO	GPIO12
D7	IO, MOSI	GPIO13
D8	IO, 10k Pull-down, SS	GPIO15
G	Ground	GND
5V	5V	-
3V3	3.3V	3.3V
RST	Reset	RST

# Project 1 : MicroPython flashen op de ESP8266

## Installatie flashtool

Om de firmware op de ESP8266 aan te passen moeten we eenmalig de flashtool esptool.py installeren. Dit is een python script dat draait op Python 2.7 en 3.4 of nieuw.

Open een terminal venster op je linux pc of Raspberry Pi en typ volgende commando's:

```
$ sudo apt-get install python-pip  
$ pip install --upgrade pip  
$ pip install --upgrade setuptools  
$ pip install --upgrade esptool
```

Om de usb poort voor het flashen toegankelijk te maken voor de ingelogde gebruiker moet deze toegevoegd worden aan de **dialout** groep.

```
$ sudo useradd -G dialout <username>
```

Reboot vervolgens de pc. De flashtool is nu geïnstalleerd.

## Installatie firmware

De officiële MicroPython firmware voor de ESP8266 kan je [hier](#) downloaden. Je kiest best de laatste stabiele versie (v1.8.7 op 01/08/2017)

```
$ mkdir micropython  
$ cd micropython  
$ wget http://micropython.org/resources/firmware/esp8266-20170108-v1.8.7.bin  
--2016-12-31 19:34:05-- http://micropython.org/resources/firmware/esp8266-20170108-v1.8.7.bin  
Herleiden van micropython.org (micropython.org)... 176.58.119.26  
Verbinding maken met micropython.org (micropython.org)|176.58.119.26|:80... verbonden.  
HTTP-verzoek is verzonden; wachten op antwoord... 200 OK  
Lengte: 565980 (553K) [application/octet-stream]  
Wordt opgeslagen als: 'esp8266-20170107-v1.8.7.bin.1'  
  
esp8266-20170108-v1.8.7.bin.1 100%  
[=====] 552,71K 2,78MB/s in 0,2s  
2016-12-31 19:34:05 (2,78 MB/s) - ''esp8266-20170108-v1.8.7.bin.1'' opgeslagen  
[565980/565980]
```

Sluit je ESP8266 bordje aan met de micro USB kabel op je PC en wis de bestaande firmware met dit commando :

```
$ esptool.py --port /dev/ttyUSB0 erase_flash  
esptool.py v1.2.1  
Connecting...  
Running Cesanta flasher stub...  
Erasing flash (this may take a while)...  
Erase took 15.0 seconds
```

Flash nu de laatste firmware op je ESP8266 bordje met dit commando :

```
$ esptool.py --port /dev/ttyUSB0 --baud 460800 write_flash --flash_size=32m 0  
esp8266-20170108-v1.8.7.bin  
esptool.py v1.2.1  
Connecting...  
Running Cesanta flasher stub...  
Flash params set to 0x0040  
Writing 569344 @ 0x0... 569344 (100 %)  
Wrote 569344 bytes at 0x0 in 13.2 seconds (344.0 kbit/s)...  
Leaving...
```

Opgelet : het ESP8266 bordje heeft 4 MB (megabyte) flashgeheugen en de –flash-size parameter moet uitgedrukt worden in megabit; vandaar de waarde 32m !

Druk na het flashen op het reset knopje van het bord, of haal even de kabel uit het bord en sluit daarna terug aan.

We installeren nog een aantal tools om te kunnen communiceren met de ESP8266 :

```
$ sudo apt-get install picocom screen
```

Probeer nu te verbinden met de seriële REPL van MicroPython met picocom of screen :

```
$ picocom /dev/ttyUSB0 -b 115200  
$ screen /dev/ttyUSB0 115200
```

en druk daarna op Enter. Als alles goed is krijg je de gekende >>> prompt van (Micro)Python en kan je interactief aan de slag.

```
>>> print('Hello world!')  
Hello world!  
>>>
```

Om picocom te sluiten gebruik je de toetscombinatie Ctrl+A gevolgd door Ctrl+X

Om screen te sluiten gebruik je de toetscombinatie Ctrl+A gevolgd door Crtl+D.

Op een windows PC kan je verbinden met Putty.

# Project 2 : Interne LED laten knipperen

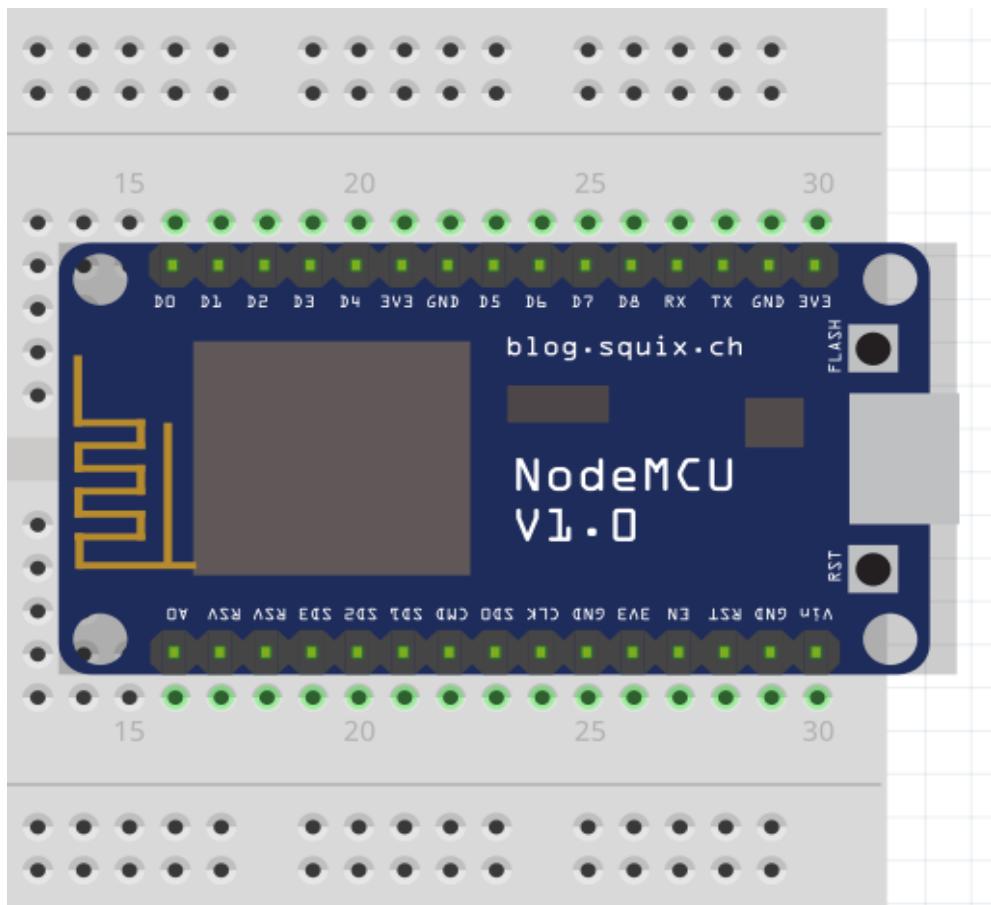
## Opgave

Laat de ingebouwde LED 10 maal knipperen met 1 seconde interval.

## Materiaal

- NodeMCU ESP8266 bordje

## Schakeling



## Code

```
from machine import Pin
import time

# interne LED zit op pin D4 (GPIO2)
PinNum = 2

pin1 = Pin(PinNum, Pin.OUT)

for t in range(10):
    pin1.high()
    time.sleep(1)
    pin1.low()
    time.sleep(1)

pin1.high()
```

# Project 3: Externe LED laten knipperen

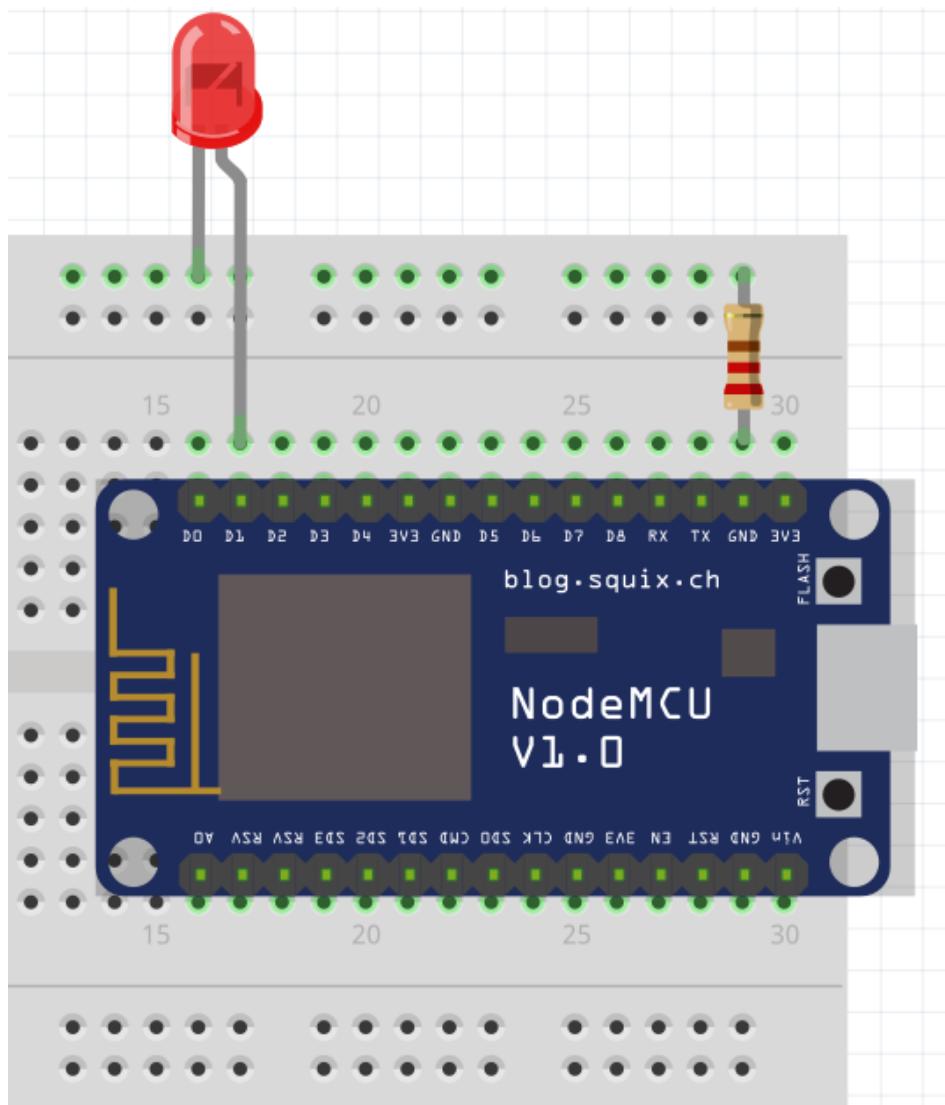
## Opgave

Laat een externe LED 10 maal knipperen met 1 seconde interval.

## Materiaal

- NodeMCU ESP8266 bordje
- LED
- weerstand 220 ohm
- jumperkabels

## Schakeling



## Code

```
from machine import Pin
import time

# externe LED zit op pin D1 (GPIO5)
PinNum = 5

pin1 = Pin(PinNum, Pin.OUT)

for t in range(10):
    pin1.low()
    time.sleep(1)
    pin1.high()
    time.sleep(1)

pin1.low()
```

# Project 4 : Faden van een externe LED met PWM

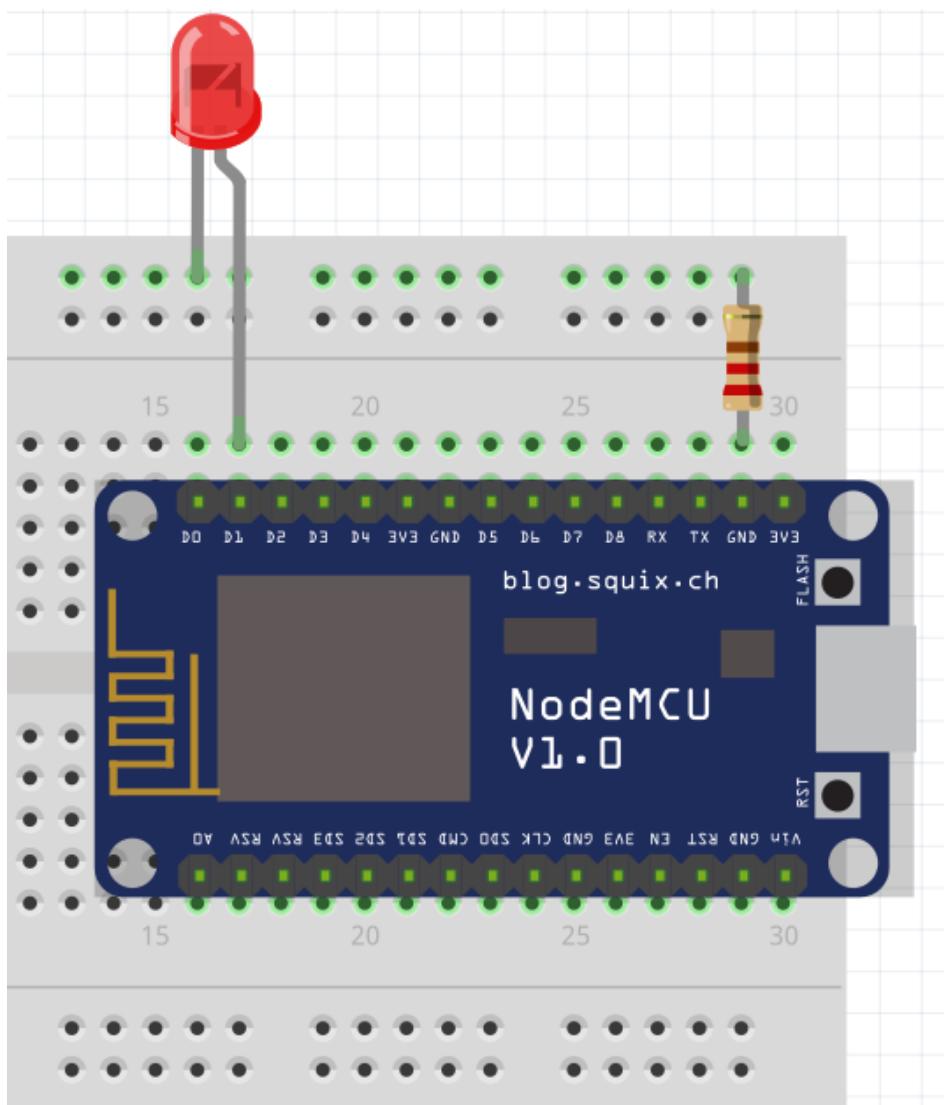
## Opgave

Laat een externe LED 10 maal langzaam feller oplichten tot zijn maximum en daarna sneller doven tot zijn minimum.

## Materiaal

- NodeMCU ESP8266 bordje
- LED
- weerstand 220 ohm
- jumperkabels

## Schakeling

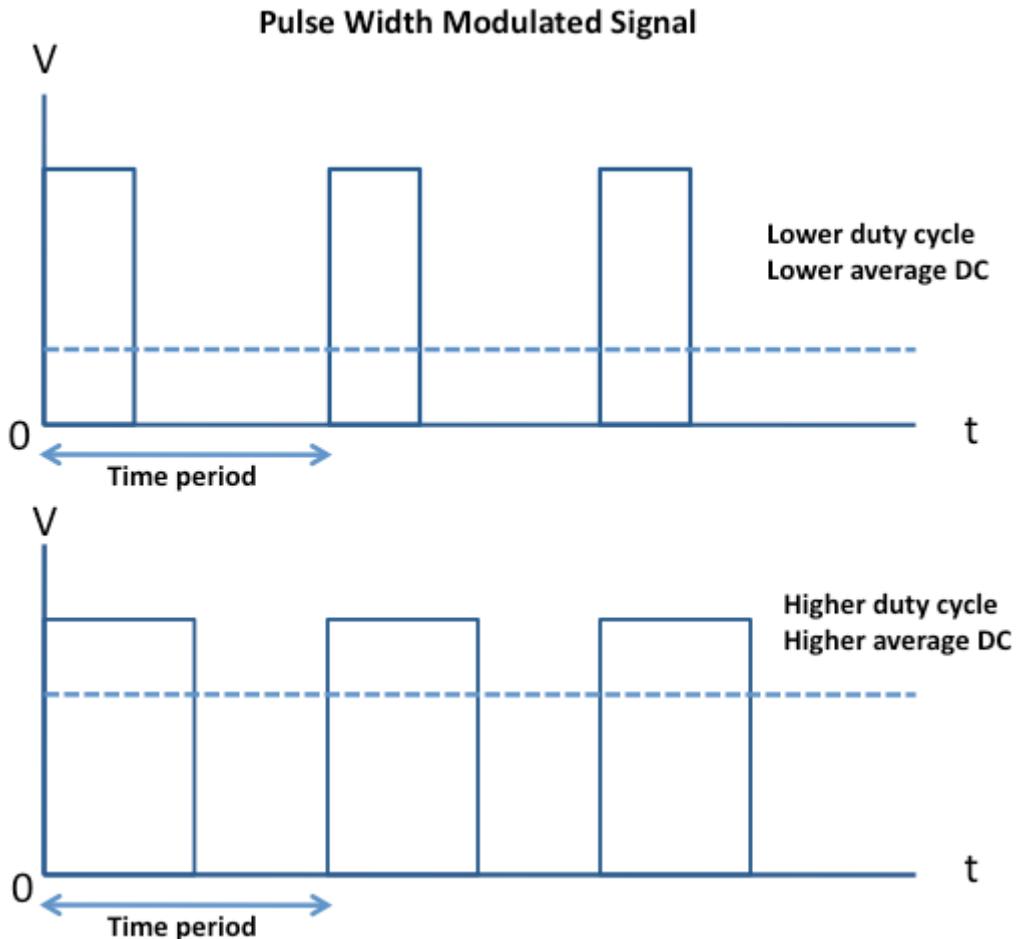


## Pulse Width Modulation

PWM is een techniek om met een digitale I/O poort (enkel hoog en laag) een analoge I/O poort (iedere waarde tussen hoog en laag) na te bootsen.

Dit wordt gedaan door een blokgolf op de digitale I/O poort te zetten. Afhankelijk van de tijd dat het signaal hoog is, spreken we van een bepaald percentage duty cycle. Hoe hoger de duty cycle, hoe hoger de gemiddelde spanning; hoe lager de duty cycle hoe lager de gemiddelde spanning.

Een PWM signaal heeft wel een beperkt aantal mogelijke waarden. Afhankelijk van de firmware is dit meestal 256 of 1024.



## Code

```
from machine import Pin, PWM
import time

# externe LED zit op pin D1 (GPIO5)
PinNum = 5

# pwm initialisatie
pwm1 = PWM(Pin(PinNum))
pwm1.freq(60)
pwm1.duty(0)

step = 100
for i in range(10):
    # oplichten
    while step < 1000:
        pwm1.duty(step)
        time.sleep_ms(500)
        step+=100
    # uitdoven
    while step > 0:
        pwm1.duty(step)
        time.sleep_ms(500)
        step-=200

# pwm resetten
pwm1.deinit()
```

# Project 5 : Schakelen van een LED met een drukknop

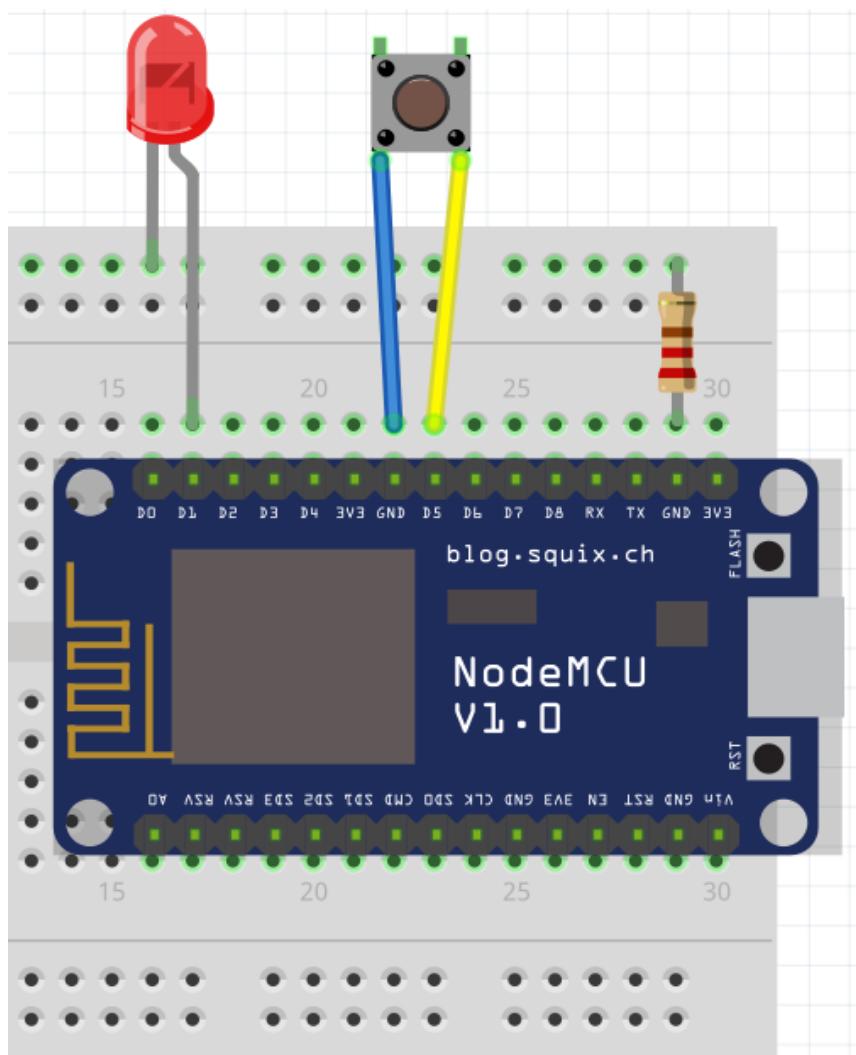
## Opgave

Schakel een externe LED aan en uit met behulp van een drukknop. We gebruiken hiervoor 2 jumperkabeltjes en de ingebouwde pull up weerstand van het poort op het bordje.

## Materiaal

- NodeMCU ESP8266 bordje
- LED
- weerstand 220 ohm
- jumperkabels

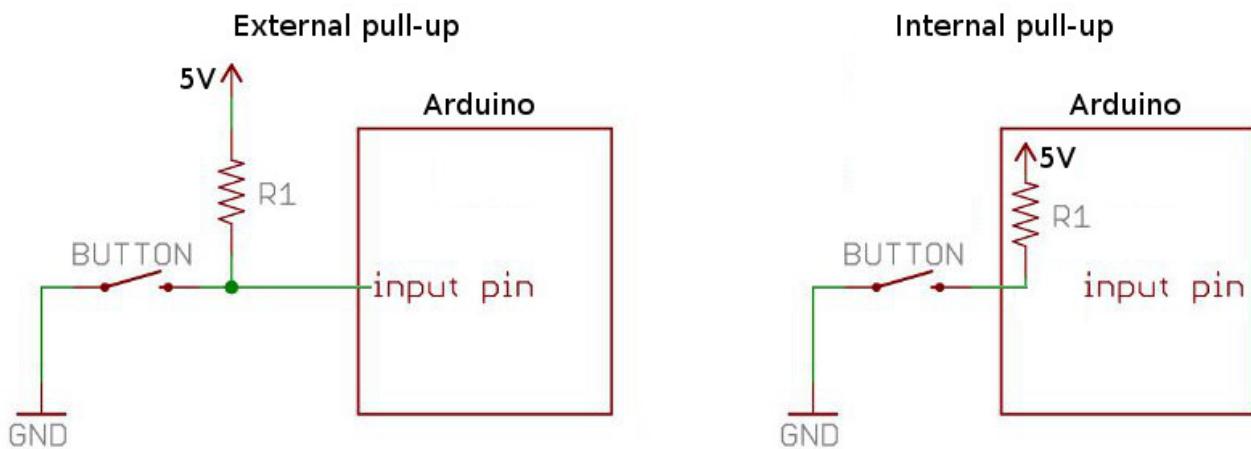
## Schakeling



## Pull up – pull down weerstand

In bovenstaande schakeling activeren we de ingebouwde pull up weerstand van poort D5. Doen we dit niet, dan zal de waarde op deze poort in een **zwevende toestand** zijn wanneer de drukknop open is. De waarde kan zowel hoog als laag zijn. Met de ingebouwde pull up wordt de poort inwendig verbonden met Vcc en zal de waarde dus hoog zijn. Bij het indrukken van de drukknop wordt de poort verbonden met GND en wordt de poort laag. Door de pull up zal de stroom ook beperkt worden zodat de poort niet beschadigd raakt.

Een pull down weerstand heeft dezelfde maar omgekeerde werking. De poort wordt inwendig via de pull down weerstand verbonden met GND. Normaal is de poort dus laag. De drukknop is nu verbonden met VCC en bij het indrukken wordt de poort hoog.



## Code

```
from machine import Pin
import time

# externe LED zit op pin D1 (GPIO5)
PinNum = 5
# button zit op pin D5 (GPIO14)
ButNum = 14

led = Pin(PinNum, Pin.OUT)
# we gebruiken de ingebouwde pullup weerstand
button = Pin(ButNum, Pin.IN, Pin.PULL_UP)

while True:
    if not button.value():
        # toggle functie
        led.value(not led.value())
        time.sleep_ms(300)
        while not button.value():
            pass
```

# Project 6 : Sturen van een servomotor

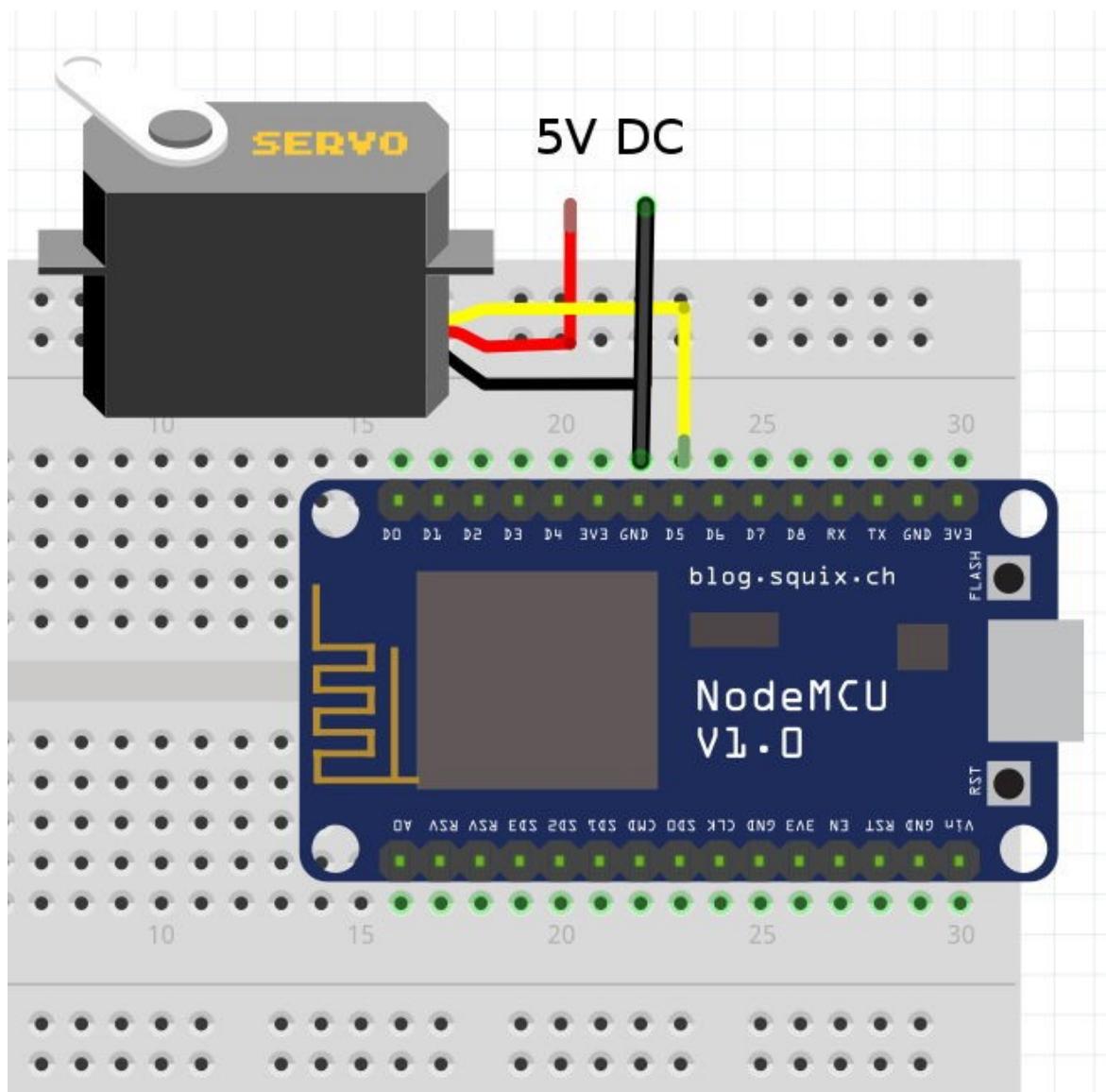
## Opgave

Laat een servo motor 10 maal van  $0^\circ$  naar  $90^\circ$  bewegen.

## Materiaal

- NodeMCU ESP8266 bordje
- Tower Pro MG90S servomotor
- Externe voeding 5V (ESP8266 levert niet genoeg stroom voor servomotor)
- jumperkabels

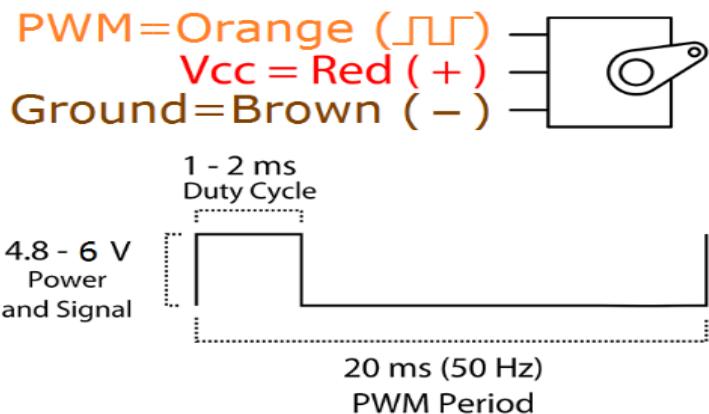
## Schakeling



## Servomotor

De servomotor wordt aangestuurd met een PWM signaal van 50 Hz. De duty cycle bepaalt de positie van de servo.

- Links : 1 ms
- Midden : 1.5 ms
- Rechts : 2 ms



## Code

```
from machine import Pin, PWM
import time

# servo motor sturing zit op pin D5 (GPI014)
PinNum = 14

# posities servo
links = 40
midden = 77
rechts = 110

# initialisatie servo
servo = PWM(Pin(PinNum), freq=50, duty=midden)

for t in range(10):
    # links
    servo.duty(links)
    time.sleep(1)
    # midden
    servo.duty(midden)
    time.sleep(1)
    # rechts
    servo.duty(rechts)
    time.sleep(1)
    # midden
    servo.duty(midden)
    time.sleep(1)

# pwm resetten
servo.duty(midden)
servo.deinit()
```

# Project 7 : Sturen van een stappenmotor met de ULN2003A

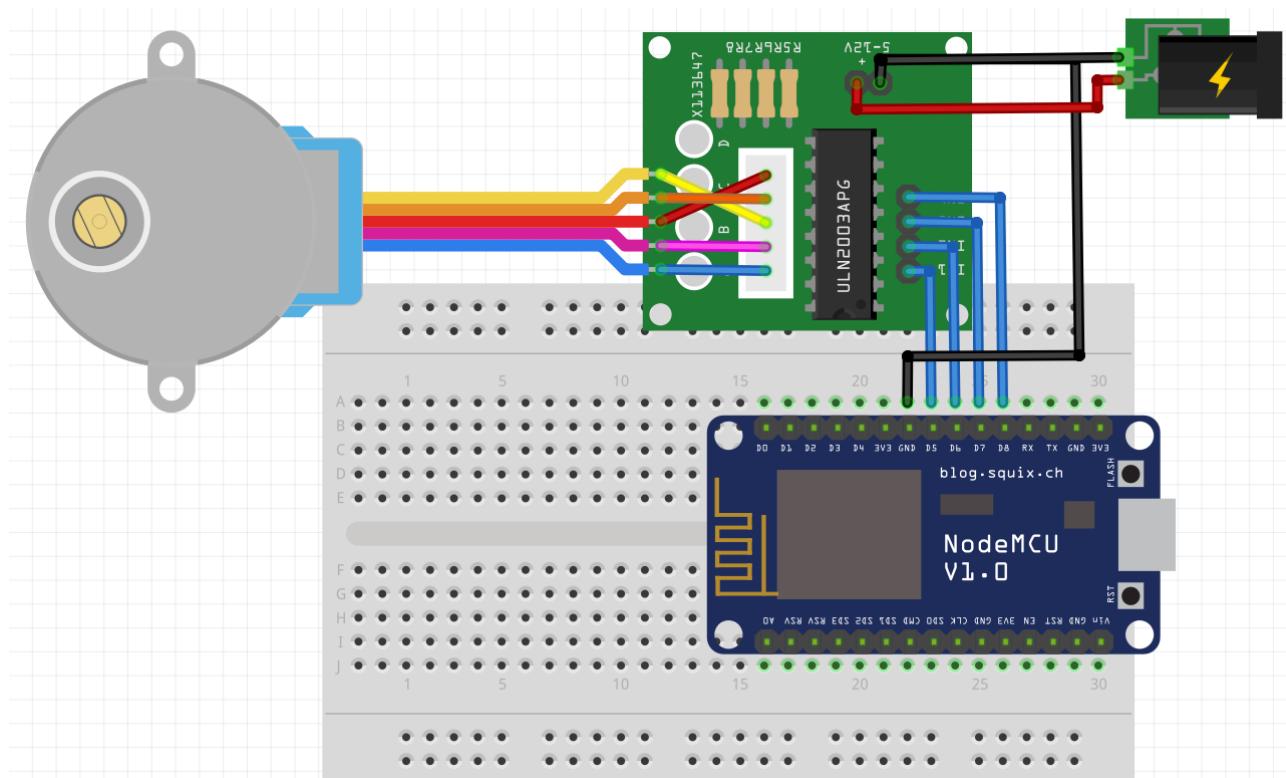
## Opgave

Gebruik de Darlington array ULN2003A om een stappenmotor eerst in wijzerzin en vervolgens in tegenwijzerzin te laten draaien.

## Materiaal

- NodeMCU ESP8266 bordje
- Stappenmotor 28BYJ-48 met ULN2003A sturingsmodule
- Externe voeding 5V
- jumperkabels

## Schakeling



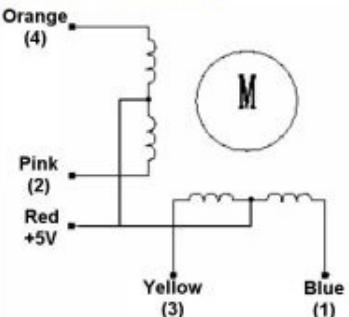
ULN2003A Pin	ESP8266 Pin	ESP8266 GPIO
IN1	D5	14
IN2	D6	12
IN3	D7	13
IN4	D8	15

## Stappenmotor

De 28BYJ-48 stappenmotor heeft 4 wikkelingen die met een bepaalde sequentie moeten hoog en laag gezet worden om de motor te doen draaien. In onderstaande tabel is de halve stap sequentie afgebeeld die uit 8 stappen bestaat. De motor draait in wijzerzin als de stappen oplopend (1 tot 8) en in tegenwijzerzin als de stappen aflopend (8 tot 1) worden uitgevoerd.

Er bestaat ook een gehele stap sequentie die maar uit 4 stappen bestaat. Hierbij worden enkel de stappen 1, 3, 5 en 7 uit de tabel gebruikt.

**WIRING DIAGRAM**



**Switching Sequence**

Lead Wire Color	---> CW Direction (1-2 Phase)							
	1	2	3	4	5	6	7	8
4 Orange	-	-						-
3 Yellow		-	-	-				
2 Pink				-	-	-		
1 Blue						-	-	-

## Code

### stepper.py

```
import time

# klasse voor stepper motor met 4 fazen + common (halve stap sequentie)
# stepper 28BYJ-48 met ULN2003A driver board
class FullCycle0YPB(object):

    def __init__(self, pins):
        self.pinColors = {"org":pins[0], "yel":pins[1], "pik":pins[2], "blu":pins[3]}
        self.pinOrder = ["org", "yel", "pik", "blu"]
        self.seqs = []

    def Build(self):
        seq = {"org":1, "yel":0, "pik":0, "blu":0}
        self.seqs.append(seq)
        seq = {"org":1, "yel":1, "pik":0, "blu":0}
        self.seqs.append(seq)
        seq = {"org":0, "yel":1, "pik":0, "blu":0}
        self.seqs.append(seq)
        seq = {"org":0, "yel":1, "pik":1, "blu":0}
        self.seqs.append(seq)
        seq = {"org":0, "yel":0, "pik":1, "blu":0}
        self.seqs.append(seq)
        seq = {"org":0, "yel":0, "pik":1, "blu":1}
        self.seqs.append(seq)
        seq = {"org":0, "yel":0, "pik":0, "blu":1}
        self.seqs.append(seq)
        seq = {"org":1, "yel":0, "pik":0, "blu":1}
        self.seqs.append(seq)
```

```

# klasse voor stepper motor met 4 fazen+common (gehele stap sequentie)
# stepper 28BYJ-48 met ULN2003A driver board
class SimpleCycleOYPB(object):

    def __init__(self, pins):
        self.pinColors = {"org":pins[0], "yel":pins[1], "pik":pins[2],
"blu":pins[3]}
        self.pinOrder = ["org", "yel", "pik", "blu"]
        self.seqs = []

    def Build(self):
        seq = {"org":1, "yel":0, "pik":0, "blu":0}
        self.seqs.append(seq)
        seq = {"org":0, "yel":1, "pik":0, "blu":0}
        self.seqs.append(seq)
        seq = {"org":0, "yel":0, "pik":1, "blu":0}
        self.seqs.append(seq)
        seq = {"org":0, "yel":0, "pik":0, "blu":1}
        self.seqs.append(seq)

# klasse om waarde van de pins op te halen en draairichting te bepalen
class Stepper(object):

    def __init__(self, delay, cycle):
        self.__delay = delay          # tijd tussen seq
        self.__seqs = cycle.seqs      # seq tabel
        self.__pins = cycle.pinOrder  # volgorde kleuren
        self.__cw = True               # wijzerzin
        self.__seqPos = 0              # positie in seq
        self.__pinPos = 0              # positie pin (kleur)
        self.__seqLen = len(self.__seqs) # aantal seq

    def setCW(self, cw=True):       # default wijzerzin
        self.__cw = cw

    def setDelay(self, delay):
        self.__delay = delay

    def nextPin(self):
        pinColor = ""                # init pinColor
        try:
            # kleur van de pin
            pinColor = self.__pins[self.__pinPos]
        except:
            time.sleep(self.__delay)    # delay voor volgende seq
            self.__pinPos = 0           # positie eerste pin
            pinColor = self.__pins[0]    # waarde eerste pin
            if self.__cw == True:       # wijzerzin
                self.__seqPos += 1       # positie + 1
            else:                      # tegenwijzerzin
                self.__seqPos -= 1       # positie - 1
            if self.__seqPos >= self.__seqLen: # test overflow
                self.__seqPos = 0
            if self.__seqPos < 0:       # test underflow positie
                self.__seqPos = self.__seqLen - 1
        # waarde van de pin
        val = self.__seqs[self.__seqPos][pinColor]
        self.__pinPos += 1             # pin positie ophogen
        return [pinColor, val]

```

### **teststepper.py**

```
from machine import Pin
from stepper import FullCycle0YPB, Stepper
import time

# initialisatie pinnen
pins = [ 14, 12, 13, 15]
in1 = Pin(14, Pin.OUT)      # pin D5
in2 = Pin(12, Pin.OUT)      # pin D6
in3 = Pin(13, Pin.OUT)      # pin D7
in4 = Pin(15, Pin.OUT)      # pin D8

# initialisatie stappenmotor
c = FullCycle0YPB(pins)
c.Build()
s = Stepper(0.008,c)

# lus draaien wijzer- en tegenwijzerzin
richting = [ True, False ]
for r in richting:
    s.setCW(r)
    for k in range(20000):
        res = s.nextPin()
        pin = res[0]
        if pin == "org":
            in1.value(res[1])
        elif pin == "yel":
            in2.value(res[1])
        elif pin == "pik":
            in3.value(res[1])
        elif pin == "blu":
            in4.value(res[1])
        time.sleep(0.0001)
```

# Project 8 : Sturen van een DC motor met een H-bridge

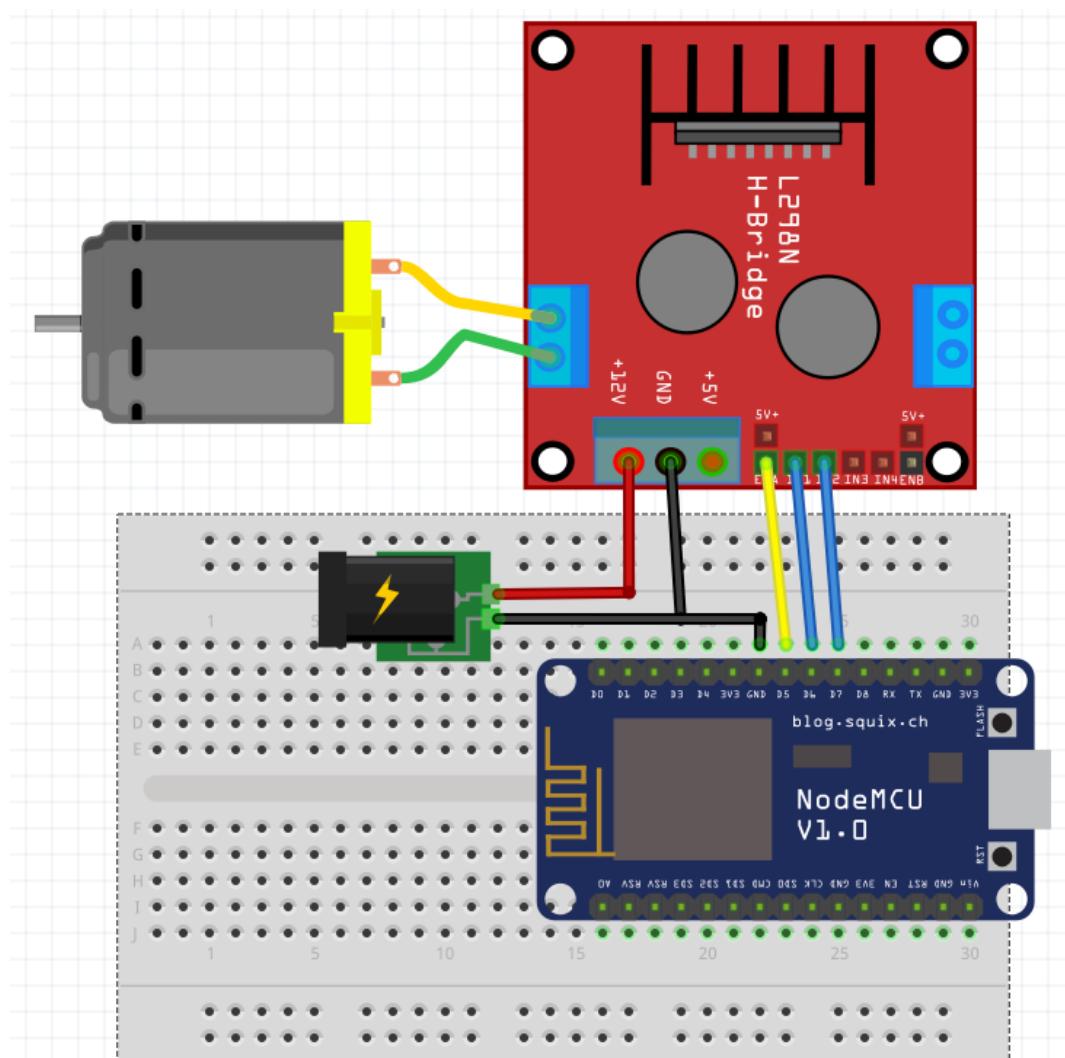
## Opgave

Gebruik de L298N H-bridge module om een gelijkstroommotor te versnellen tot aan zijn maximum en vervolgens te vertragen tot volledige stilstand. Doe dit voor beide richtingen.

## Materiaal

- NodeMCU ESP8266 bordje
- DC motor en L298N stuurmodule
- Externe voeding 5V
- jumperkabels

## Schakeling



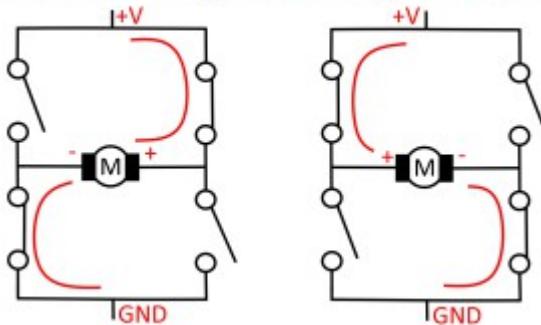
L298N Pin	ESP8266 Pin	ESP8266 GPIO
ENA	D5	14
IN1	D6	12
IN2	D7	13

## H-bridge

Een DC motor heeft slechts 2 aansluitingen.

Afhankelijk van hoe de spanning wordt aangesloten draait de motor in wijzer- of tegenwijzerzin. Om dit mogelijk te maken wordt een H-bridge schakeling gebruikt. Dit is eigenlijk een geheel van 4 Darlington transistor schakelaars die de spanning op de wikkeling van de motor kunnen omkeren. De L298 is een dubbele H-bridge en kan dus 2 DC motoren aansturen.

How an H-bridge can change direction



## Code

### dcmotor.py

```
from machine import Pin, PWM

class HBridge:
    """ Klasse H-Bridge voor het aansturen van 1 gelijkstroom motor.
        Laat de motor voorwaarts of achterwaarts draaien.
        Gebruikt PWM om de snelheid van de motor te varieren van 0 tot 100%.
        Als geen PWM gebruikt wordt is de maximale snelheid steeds 100% """
    # default pwm frequentie en duty cycle
    PWM_FREQ = 60          # in Hz
    PWM_DUTY = 0            # in %

    # status motor
    (ONBEKEND, STOPPEN, VOORUIT, ACHTERUIT) = (-1, 0, 1, 2)

    # initialisatie H-Bridge
    def __init__(self, input_pins, pwm_pin=None):
        """param input_pins: tuple met de 2 stuурpinnen
           :param pwm_pin: PWM pin voor de snelheid (None=geen PWM) """
        self.snelheid = 0
        self.status = HBridge.ONBEKEND
        # Initialisatie HBridge stuurrpinnen
        self.in1 = Pin(input_pins[0], Pin.OUT)
        self.in2 = Pin(input_pins[1], Pin.OUT)
        # Initialisatie PWM Enable pin voor snelheidscontrole
        # zonder PWM moet de Enable pin van de L293D hoog staan
        self.met_pwm = (pwm_pin is not None)
        if self.met_pwm:
            self.ena = PWM(Pin(pwm_pin, Pin.OUT))
            self.ena.freq(self.PWM_FREQ)
            self.ena.duty(self.pwm_waarde(self.PWM_DUTY))
```

```

# Alles stoppen
self.stoppen()

def pwm_waarde(self, snelheid):
    # omzetten snelheid in % naar PWM waarde (0-1023)
    return int(snelheid / 100.0 * 1023.0)

def zet_snelheid(self, snelheid):
    if not(0 <= snelheid <= 100):
        raise ValueError('Ongeldige snelheid')
    if self.met_pwm:                      # snelheid met PWM
        self.ena.duty(self.pwm_waarde(snelheid))
        self.snelheid = snelheid
    else:                                  # zonder PWM
        if snelheid == 0:
            self.snelheid = 0
        else:
            self.snelheid = 100
    if self.snelheid == 0 and self.status != HBridge.STOPPEN:
        self.stoppen()                  # motor stoppen

def stoppen(self):
    self.in1.low()                      # in1 laag
    self.in2.low()                      # in2 laag
    self.status = HBridge.STOPPEN      # Deze 2 lijnen ...
    self.zet_snelheid(0)                # niet omkeren

def vooruit(self, snelheid=100):
    # HBridge herinstellen
    if self.status != HBridge.VOORUIT:
        self.stoppen()
        self.in1.low()                  # in1 laag
        self.in2.high()                 # in2 hoog
        self.status = HBridge.VOORUIT
    # snelheid instellen
    self.zet_snelheid(snelheid)

def achteruit(self, snelheid=100):
    # HBridge herinstellen
    if self.status != HBridge.ACHTERUIT:
        self.stoppen()
        self.in1.high()                 # in1 hoog
        self.in2.low()                  # in2 laag
        self.status = HBridge.ACHTERUIT
    # snelheid instellen
    self.zet_snelheid(snelheid)

class DualHBridge():
    """ klasse Dual H-Bridge voor het aansturen van 2 gelijkstroom motoren.
    Laat de motoren voorwaarts of achterwaarts draaien.
    Gebruikt PWM om de snelheid van de motoren te variëren van 0 tot 100%.
    Als geen PWM gebruikt wordt is de maximale snelheid steeds 100% """
    def __init__(self, mot1_pins, mot1_pwm, mot2_pins, mot2_pwm, afwijkking=0):
        """param mot1_pins: tuple met de 2 stuурpinnen voor motor 1
        :param mot1_pwm: PWM pin voor snelheid van motor 1 (None=geen PWM)
        :param mot2_pins: tuple met de 2 stuurrpinnen voor motor 2
        :param mot2_pwm: PWM pin voor snelheid van motor 2 (None=geen PWM)
        :param afwijkking: +3 om motor 1 te versnellen bij vooruit rijden
                           -3 om motor 1 te vertragen bij vooruit rijden"""

```

```

        if (afwijking != 0) and ((mot1_pwm is None) or (mot2_pwm is None)):
            raise ValueError('Afwijking instelling werkt alleen met PWM')
        self.motor1 = HBridge(mot1_pins, mot1_pwm)
        self.motor2 = HBridge(mot2_pins, mot2_pwm)
        self.afwijking = afwijking

    def vooruit(self, snelheid=100, snelheid_2=None):
        if snelheid_2 is None:                      # snelheid motor 2
            snelheid_2 = snelheid                  # zelfde snelheid als motor 1
        # aanpassen snelheid motor1 met afwijking
        if self.afwijking > 0:                      # afwijking positief
            snelheid = snelheid - self.afwijking
            if snelheid < 0:                         # snelheid niet negatief !
                snelheid = 0
        elif self.afwijking < 0:                      # afwijking negatief
            snelheid_2 = snelheid_2 - abs(self.afwijking)
            if snelheid_2 < 0:
                snelheid_2 = 0                      # snelheid niet negatief
        # motoren aansturen
        self.motor1.vooruit(snelheid)
        self.motor2.vooruit(snelheid_2)

    def achteruit(self, snelheid=100, snelheid_2=None):
        if speed_2 is None:                        # snelheid motor 2
            speed_2 = speed                      # zelfde snelheid als motor 1
        # aanpassen snelheid motor1 met afwijking
        if self.afwijking > 0:                      # afwijking positief
            snelheid = snelheid - self.afwijking
            if snelheid < 0:                         # snelheid niet negatief !
                snelheid = 0
        elif self.afwijking < 0:                      # afwijking negatief
            snelheid_2 = snelheid_2 - abs(self.afwijking)
            if snelheid_2 < 0:
                snelheid_2 = 0                      # snelheid niet negatief
        # motoren aansturen
        self.motor1.achteruit(snelheid)
        self.motor2.achteruit(snelheid_2)

    def stoppen(self):
        self.motor1.stoppen()
        self.motor2.stoppen()

```

### ***test\_dcmotor.py***

```
from dcmotor import HBridge
import time

# pinnen HBridge L293
IN1 = 12          # pin D6
IN2 = 13          # pin D7
ENA = 14          # pin D5

# motorpinnen initialiseren
motor = HBridge([IN1, IN2], ENA)

# motor voorwaarts versnellen tot maximum
snelheid = 0
while snelheid < 100:
    print("F {} %".format(snelheid))
    motor.vooruit(snelheid)
    time.sleep_ms(250)
    snelheid+=1

# motor voorwaarts vertragen tot minimum
while snelheid > 0:
    print("F {} %".format(snelheid))
    motor.vooruit(snelheid)
    time.sleep_ms(250)
    snelheid-=1

# motor achterwaarts versnellen tot maximum
snelheid = 0
while snelheid < 100:
    print("B {} %".format(snelheid))
    motor.achteruit(snelheid)
    time.sleep_ms(250)
    snelheid+=1

# motor achterwaarts vertragen tot minimum
while snelheid > 0:
    print("B {} %".format(snelheid))
    motor.achteruit(snelheid)
    time.sleep_ms(250)
    snelheid-=1

# motor stoppen
motor.stoppen()
```

# Project 9 : Een melodietje afspelen op een speaker

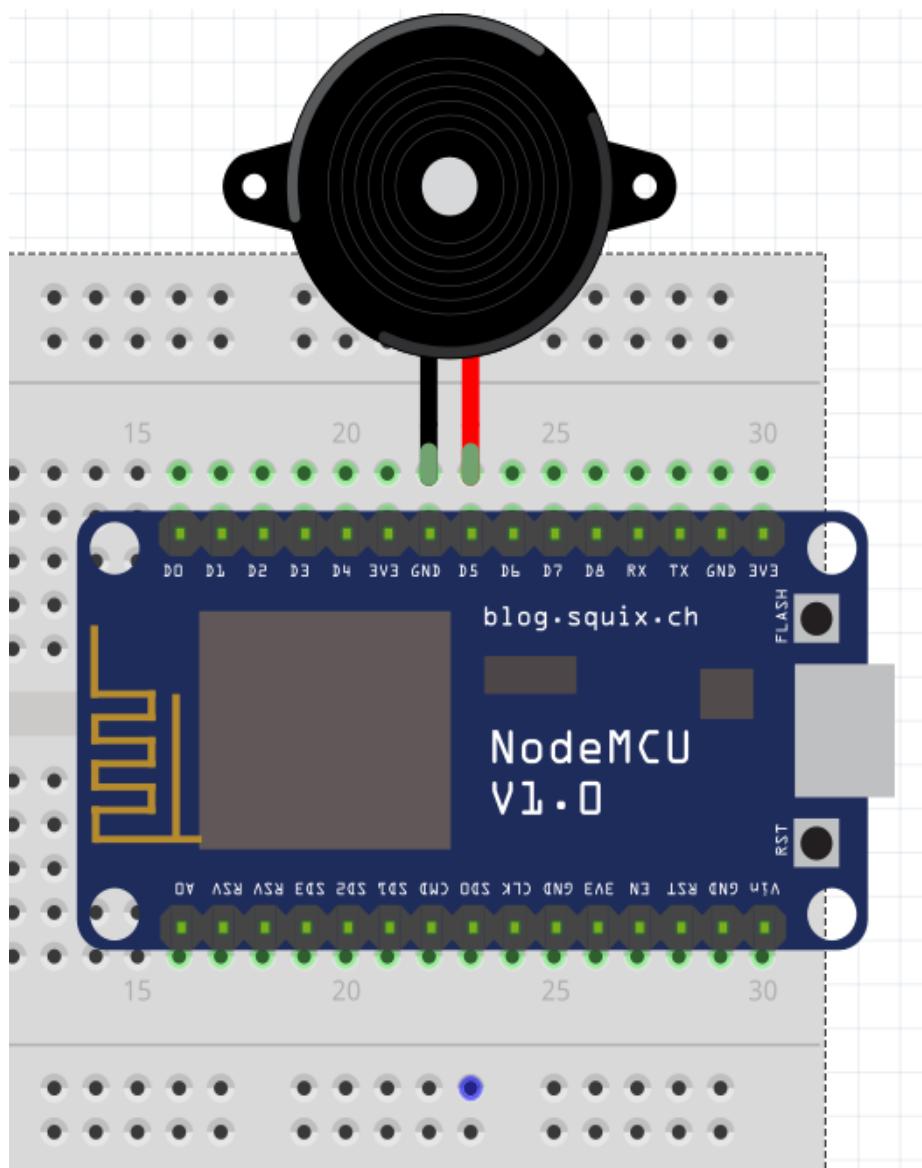
## Opgave

Koppel een speaker aan je ESP en experimenteer wat met geluid. Gebruik hiervoor een pwm signaal waarvan je de frequentie laat variëren.

## Materiaal

- NodeMCU ESP8266 bordje
- piëzo speaker
- jumperkabels

## Schakeling



## Piëzo

Een piëzo zoemer is een elektronisch component die elektrische energie omzet in mechanische energie in de vorm van trillingen. Er ontstaan dan sinusvormige geluidsgolven die typisch in de hoorbare frequenties van 20 Hz tot 20 kHz vallen. We gebruiken hiervoor uiteraard PWM signalen waarvan we de frequentie laten variëren.

Het omgekeerde effect waarbij een mechanische kracht op het piëzo element elektrische energie genereert bestaat ook. Dat wordt onder andere gebruikt voor de detectie van schokken.

## Code

```
from machine import Pin, PWM
import time

# piezo speaker zit op pin D5 (GPIO14)
PinNum = 14

# constanten
tempo = 5
tones = {
    'c': 262,
    'd': 294,
    'e': 330,
    'f': 349,
    'g': 392,
    'a': 440,
    'b': 494,
    'C': 523,
    ' ': 0,
}
melody = 'cdefgabC'
rhythm = [8, 8, 8, 8, 8, 8, 8, 8]

# initialisatie beeper
beeper = PWM(Pin(14, Pin.OUT), freq=440, duty=512)

# speel toonladder
for tone, length in zip(melody, rhythm):
    beeper.freq(tones[tone])
    time.sleep(tempo/length)

# reset beeper
beeper.deinit()
```

# Project 10 : Uitlezen van een DH11/DT22 sensor

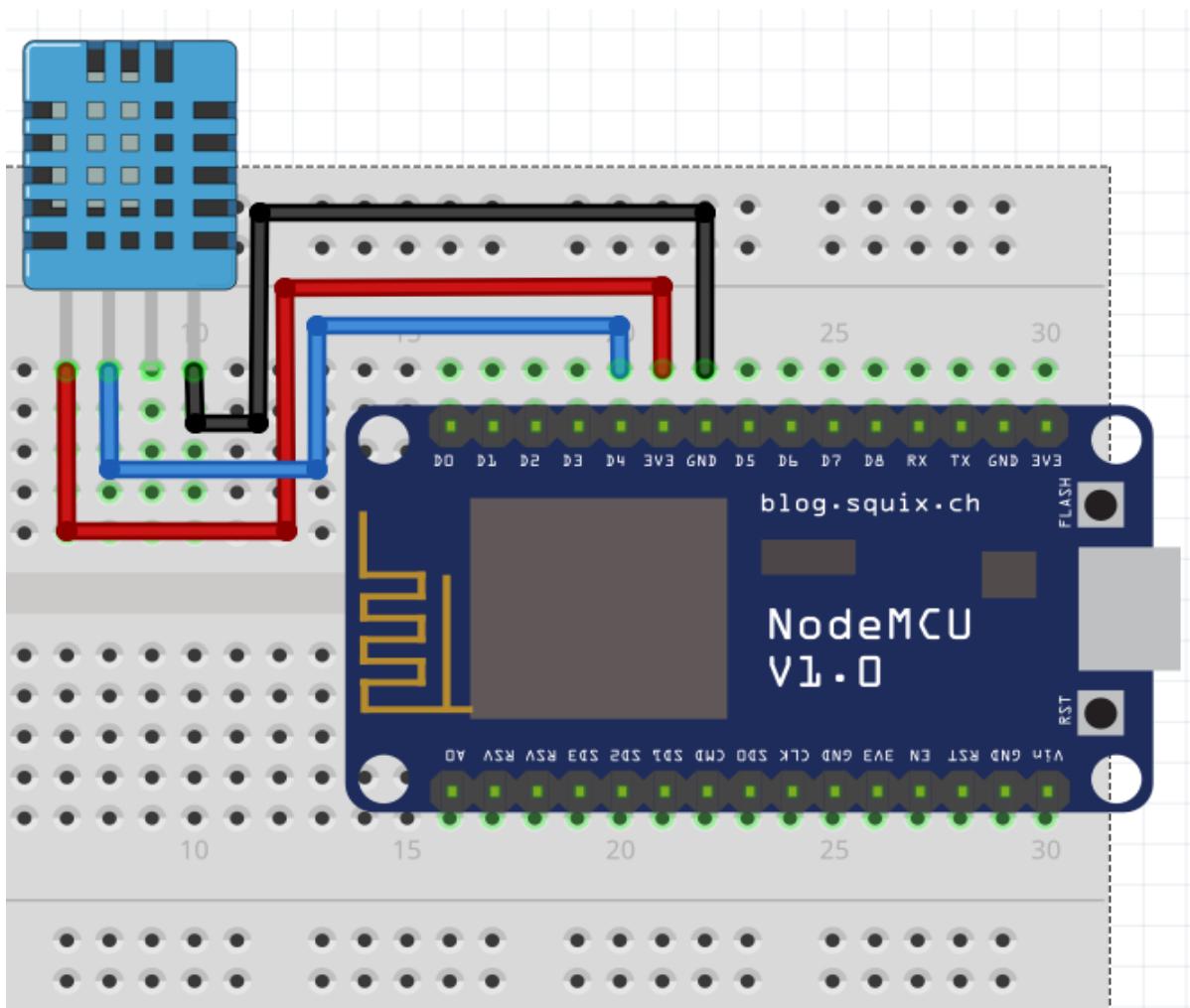
## Opgave

Vraag om de 10 seconden de vochtigheid en temperatuur op en druk het volgende af in de REPL:  
dag-maand-jaar uur:min:sec – temperatuur °C – vochtigheid %.

## Materiaal

- NodeMCU ESP8266 bordje
- DT11 of DT22 sensor
- jumperkabels

## Schakeling



DHT11 Pin	ESP8266 Pin	ESP8266 GPIO
VDD	3V3	-
DATA	D4	2
NC	-	-
GND	GND	-

## Code

```
from machine import Pin
import dht
import time
import utime

# datapin van DHT11 zit op pin D4 (GPIO2)
DataPin = 2

# initialisatie DHT11 sensor
sensor = dht.DHT11(Pin(DataPin))

while True:
    # lees sensorwaarde
    sensor.measure()

    # lees tijd (geen RTC !)
    tijd = utime.localtime()

    # toon waarde
    print("{:2d}/{:2d}/{:4d} {:2d}:{:2d}:{:2d} - {:.1f}C - {:.1f}%
".format(tijd[2], tijd[1], tijd[0], tijd[3], tijd[4], tijd[5],
sensor.temperature(), sensor.humidity()))

    # wacht 10 sec
    time.sleep(10)
```

## REPL output

```
>>> import DHT11
1/ 1/2000 0: 4: 0 - 22.0C - 36.0%
1/ 1/2000 0: 4: 9 - 23.0C - 41.0%
1/ 1/2000 0: 4:19 - 24.0C - 51.0%
1/ 1/2000 0: 4:28 - 27.0C - 80.0%
1/ 1/2000 0: 4:38 - 27.0C - 91.0%
1/ 1/2000 0: 4:47 - 28.0C - 90.0%
1/ 1/2000 0: 4:57 - 28.0C - 88.0%
1/ 1/2000 0: 5: 6 - 28.0C - 79.0%
1/ 1/2000 0: 5:15 - 28.0C - 70.0%
1/ 1/2000 0: 5:25 - 33.0C - 65.0%
1/ 1/2000 0: 5:34 - 27.0C - 66.0%
1/ 1/2000 0: 5:44 - 26.0C - 66.0%
1/ 1/2000 0: 5:53 - 25.0C - 65.0%
```

# Project 11 : Uitlezen analoge poort van de ESP8266

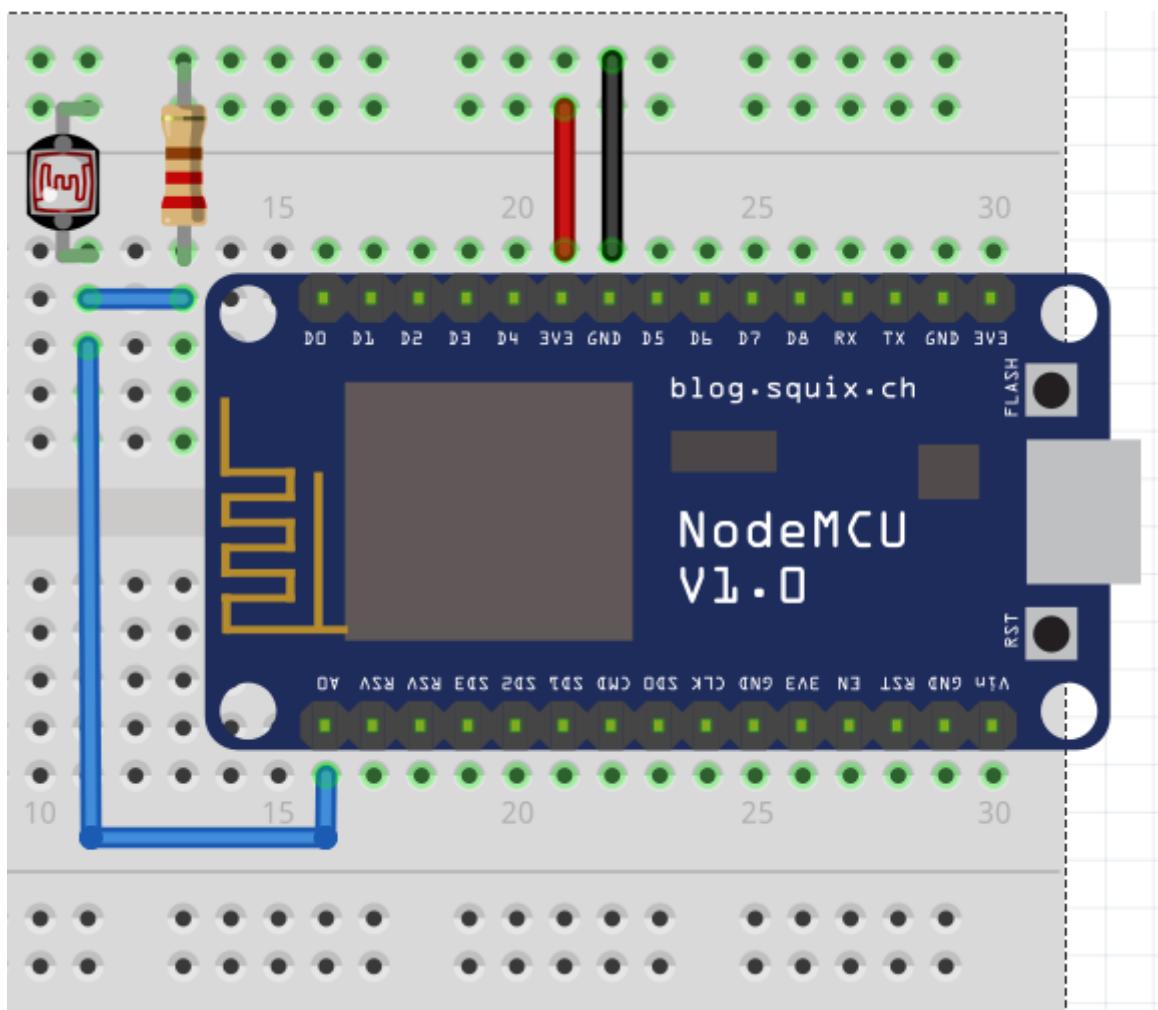
## Opgave

Lees de waarde van een lichtgevoelige weerstand (LDR) uit met de analoge poort A0 van de ESP8266. Opgelet: de spanning op de analoge poort mag **maximaal 1 Volt** bedragen !

## Materiaal

- NodeMCU ESP8266 bordje
- LDR (Light Dependent Resistor)
- aangepaste weerstand (zie verder)
- jumperkabels

## Schakeling

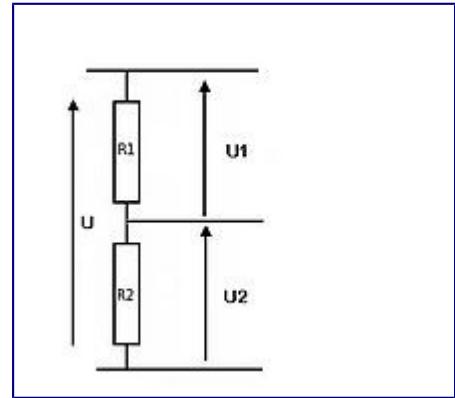


## Spanningsdeler

Om zeker te zijn dat de spanning op de analoge poort A0 niet hoger is dan 1V maken we gebruik van een spanningsdeler. De weerstand van de LDR bij minimale belichting is ongeveer 50K ohm en bij maximale belichting ongeveer 50 ohm. **Meet dit uit met je voltmeter !!!**

Formule :  $U_2 = U * (R_2 / (R_1 + R_2))$

Of in mensentaal : de totale spanning verdeelt zich over de weerstanden als de verhouding van hun waarden. Als  $U_2$  maximaal 1 V mag zijn, dan moet er in dat geval 2,3 V over  $R_1$  (LDR) staan. De verhouding is dus 1:2,3. Bij maximale belichting is de weerstand van de LDR ongeveer 50 ohm en dus moet  $R_2$  50 / 2,3 of ongeveer 20 ohm zijn.



## Code

```
from machine import ADC
import time

# initialisatie analoge poort A0
adc = ADC(0)

while True:
    # toon uitgelezen waarde
    print(adc.read())

    # pause
    time.sleep_ms(500)
```

## REPL output

```
>>> import leesA0
22
17
20
27
80
149
157
109
82
111
161
158
149
136
109
22
```

# Project 12 : Analoge waarde van een ADC uitlezen via SPI

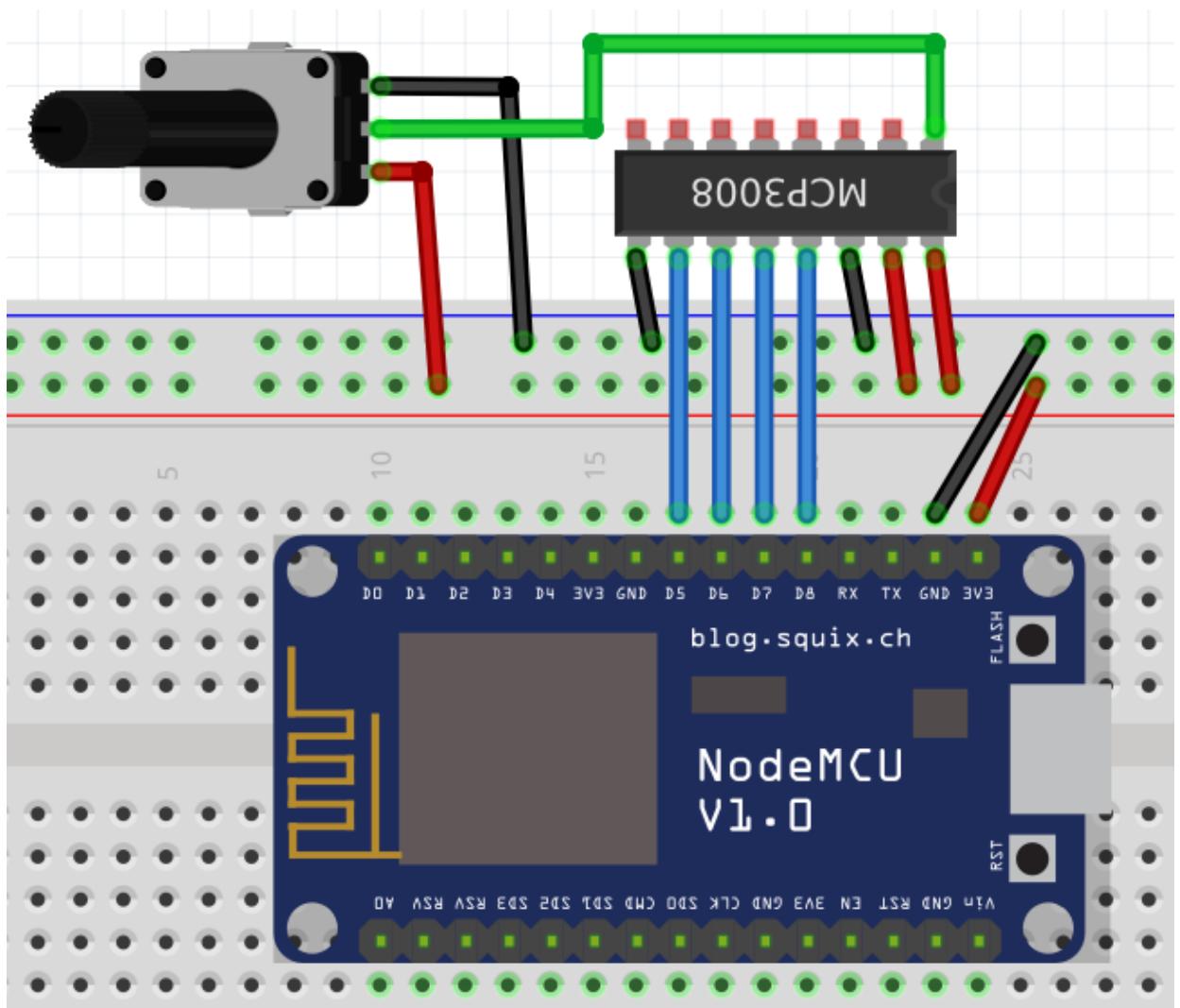
## Opgave

Lees de waarde van een regelbare weerstand met een MCP3008 ADC via software SPI en druk deze af in de REPL.

## Materiaal

- NodeMCU ESP8266 bordje
- MCP3008 Analoog Digitaal Convertor
- regelbare weerstand 10K
- jumperkabels

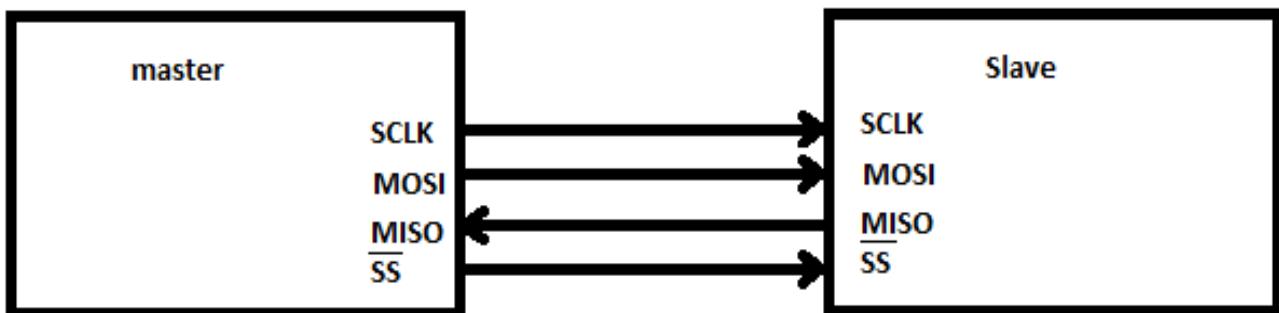
## Schakeling



MCP3008 Pin	ESP8266 Pin	ESP8266 GPIO
CH0	(slider potmeter)	-
DGND	GND	-
CS/SHDN	D5	14
DIN	D6	12
DOUT	D7	13
CLK	D8	15
AGND	GND	-
VREF	3V3	-
VDD	3V3	-

## Serial Peripheral Interface (SPI)

SPI is een synchrone seriële datalink tussen ten minste twee apparaten. Er is altijd sprake van één master en tenminste één slave. De communicatie tussen master en slave gebeurt ten allen tijde tegelijkertijd (full duplex). De start van de communicatie gebeurt door de master. Het is mogelijk om meerdere slaves te hebben, maar ze moeten dan ieder een aparte Chip Select hebben.



Er zijn altijd vier verbindingen voor communicatie nodig. Daarom spreken we van een *four-wire serial bus*.

**SCLK** : seriële clock – wordt bestuurd door de master.

**MOSI** : Master Out Slave In – hierover gaat de data van de master naar de slave.

**MISO**: Master In Slave Out – hierover gaat de data van de slave naar de master.

**SS**: Slave Select – de master zet deze lijn laag om de slave te doen luisteren. Wanneer de communicatie compleet is wordt de lijn terug hoog gezet.

Op de ESP8266 zijn 2 soorten SPI beschikbaar :

1. *Hardware SPI* : SS op GPIO15, SCLK op GPIO14, MOSI op GPIO13, MISO op GPIO12.  
Deze SPI zit ingebakken in de hardware en haalt een snelheid tot 80 MHz.
2. *Software SPI* : SS, SCLK, MOSI, MISO kunnen op iedere GPIO pin gezet worden. De implementatie gebeurt in software en is heel wat trager dan hardware SPI.

## Code

```
from machine import Pin, SPI
import time

# ADC MCP3008 met potentiometer op channel 0 pin (CH0)
# 3 bytes versturen naar DIN van MCP3008:
# 1. b0000001 : start byte
# 2. b1000xxxx : 1=single channel + 000=channel 0
# 3. bxxxxxxxx : don't care byte
DIN = bytearray([1, 128, 0])
# 3 byte antwoord van MCP3008 in DOUT:
# 1. bxxxxxxxx : niet relevant
# 2. bxxxxxx0ii : bit 9 en 8 van 10bit antwoord
# 3. biyyyyyy : bit 7 tem bit 0 van 10bit antwoord
DOUT = bytearray(3)

# initialisatie software SPI bus
# SPI      Master      Slave
# PIN      ESP8266    MCP3008
# -----
# SCK      D8 (15)    SCK
# MOSI     D6 (12)    DIN
# MISO     D7 (13)    DOUT
# CS       D5 (14)    CS
# polarity=1 -> idle state of SCK
# phase=0 -> data op stijgende flank SCK
# phase=1 -> data op dalende flank SCK
SCK = Pin(15, Pin.OUT)
MOSI = Pin(12, Pin.OUT)
MISO = Pin(13, Pin.OUT)
CS = Pin(14, Pin.OUT)
spi = SPI(-1, baudrate=100000, polarity=1, phase=0,
          sck=SCK, mosi=MOSI, miso=MISO)

# zet baudrate
spi.init(baudrate=200000)

# oneindige lus
while True:
    # zet Chip Select voor communicatie
    CS.low()
    time.sleep_us(5)
    # zend commando en lees antwoord
    spi.write_readinto(DIN, DOUT)
    # reset Chip Select
    CS.high()
    time.sleep_us(5)
    # converteer 10 LSB uit antwoord en druk af in REPL
    waarde = (((3 & DOUT[1]) << 8) + DOUT[2]) / 1023) * 3.3
    print("Spanning over potentiometer: {:.1f} V".format(waarde))
    # even wachten
    time.sleep_ms(500)
```

## ***REPL output***

```
>>> import test_adc_spi
Spanning over potentiometer: 3.3 V
Spanning over potentiometer: 0.9 V
Spanning over potentiometer: 0.8 V
Spanning over potentiometer: 0.8 V
Spanning over potentiometer: 0.0 V
Spanning over potentiometer: 0.0 V
Spanning over potentiometer: 0.0 V
Spanning over potentiometer: 0.6 V
Spanning over potentiometer: 0.9 V
Spanning over potentiometer: 1.1 V
Spanning over potentiometer: 1.7 V
Spanning over potentiometer: 1.7 V
Spanning over potentiometer: 1.9 V
Spanning over potentiometer: 2.2 V
Spanning over potentiometer: 2.2 V
Spanning over potentiometer: 2.3 V
Spanning over potentiometer: 2.6 V
Spanning over potentiometer: 2.3 V
Spanning over potentiometer: 1.8 V
Spanning over potentiometer: 1.5 V
Spanning over potentiometer: 1.7 V
Spanning over potentiometer: 2.2 V
Spanning over potentiometer: 2.3 V
Spanning over potentiometer: 2.5 V
Spanning over potentiometer: 2.6 V
Spanning over potentiometer: 2.7 V
```

# Project 13 : LED matrix aansturen via SPI

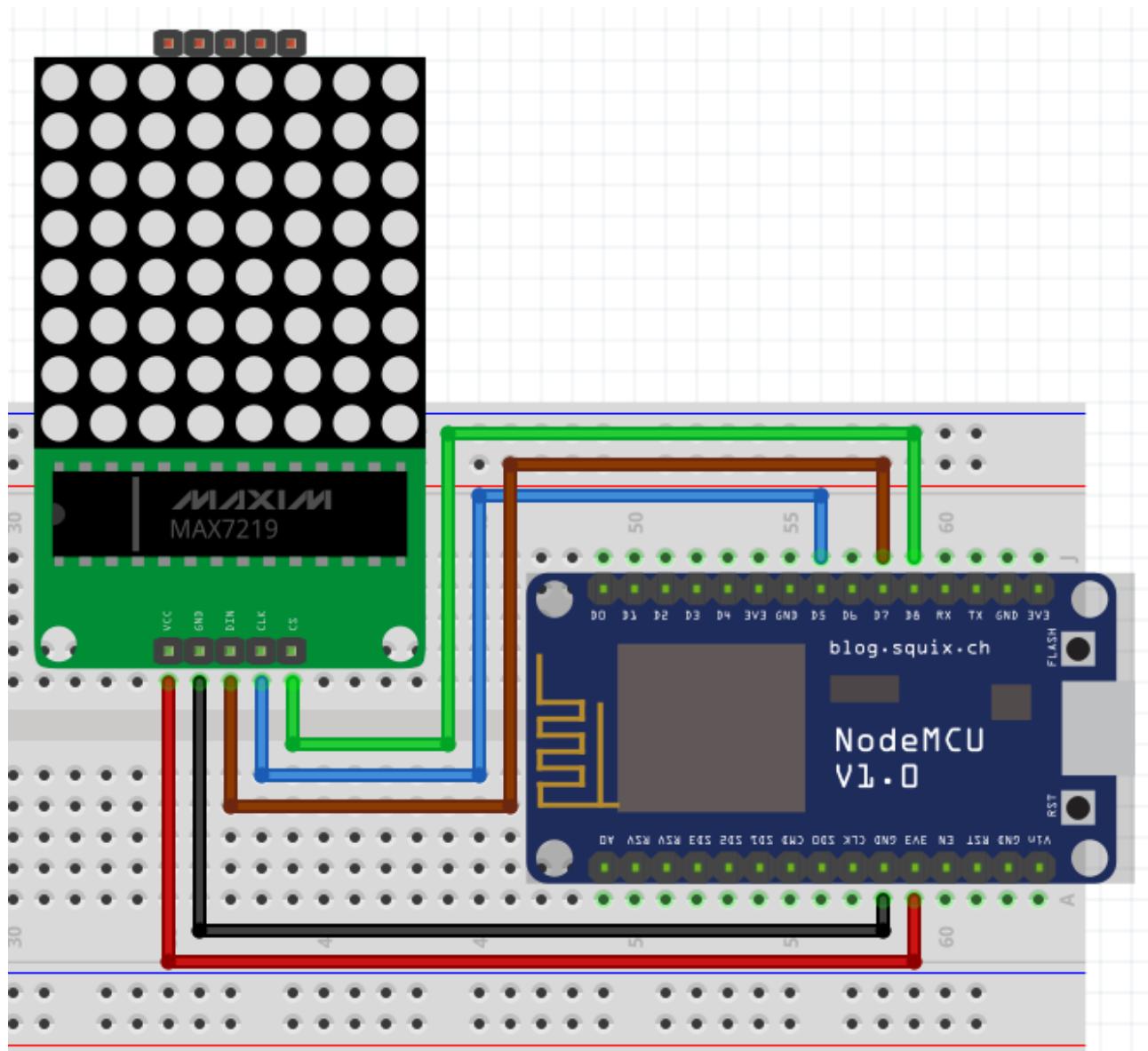
## Opgave

Laat de leds op een 8x8 LED matrix via hardware SPI één voor één oplichten.

## Materiaal

- NodeMCU ESP8266 bordje
- 8x8 LED matrix met MAX7219 SPI controller
- jumperkabels

## Schakeling



<b>MAX7219 Pin</b>	<b>ESP8266 Pin</b>	<b>ESP8266 GPIO</b>
VCC	3V3	-
GND	GND	-
DIN	D7	13
CLK	D5	14
CS	D8	15

## Code

*max7219.py*

```

_NOOP = 0x00
_DIGIT0 = 0x01
_DIGIT1 = 0x02
_DIGIT2 = 0x03
_DIGIT3 = 0x04
_DIGIT4 = 0x05
_DIGIT5 = 0x06
_DIGIT6 = 0x07
_DIGIT7 = 0x08
_DECODEMODE = 0x09
_INTENSITY = 0x0A
_SCANLIMIT = 0x0B
_SHUTDOWN = 0x0C
_DISPLAYTEST = 0x0F

class Matrix8x8:
    def __init__(self, spi, cs):
        self.spi = spi
        self.cs = cs
        self.cs.init(cs.OUT, True)
        self.buffer = bytearray(8)
        self.init()

    def _register(self, command, data):
        self.cs.low()
        self.spi.write(bytearray([command, data]))
        self.cs.high()

    def init(self):
        for command, data in (
            (_SHUTDOWN, 0),
            (_DISPLAYTEST, 0),
            (_SCANLIMIT, 7),
            (_DECODEMODE, 0),
            (_SHUTDOWN, 1),
        ):
            self._register(command, data)

```

```

def brightness(self, value):
    if not 0 <= value <= 15:
        raise ValueError("Brightness out of range")
    self._register(_INTENSITY, value)

def fill(self, color):
    data = 0xff if color else 0x00
    for y in range(8):
        self.buffer[y] = data

def pixel(self, x, y, color=None):
    if color is None:
        return bool(self.buffer[y] & 1 << x)
    elif color:
        self.buffer[y] |= 1 << x
    else:
        self.buffer[y] &= ~(1 << x)

def show(self):
    for y in range(8):
        self._register(_DIGIT0 + y, self.buffer[y])

```

### ***test\_ledmatrix.py***

```

# 8x8 LED matrix gebaseerd op de MAX7219 controller
from machine import Pin, SPI
from max7219 import Matrix8x8
import time

ON = True
OFF = False
LED_ROWS = 8
LED_COLS = 8

# initialisatie hardware SPI bus
# SPI      Master      Slave
# PIN      ESP8266    MAX7219
# -----
# SCK      D5 (14)    SCK
# MOSI     D7 (13)    DIN
# MISO     D6 (12)    ---
# CS       D8 (15)    CS
# polarity=0 -> idle state of SCK
# phase=0 -> data op stijgende flank SCK
# phase=1 -> data op dalende flank SCK
spi = SPI(1, baudrate=20000000, polarity=0, phase=0)
cs = Pin(15, Pin.OUT)

# initialisatie 8x8 LED matrix
matrix = Matrix8x8(spi, cs)
matrix.init()

# helderheid (0-15)
matrix.brightness(15)

```

```
# matrix blanko
matrix.fill(OFF)
matrix.show()

# wandelende pixel
for p in range(LED_ROWS * LED_COLS):
    x = p // LED_COLS
    y = p % LED_COLS
    # print("x:{} - y:{}".format(x, y))
    matrix.pixel(x, y, ON)
    matrix.show()
    time.sleep_ms(500)
    matrix.pixel(x, y, OFF)
    matrix.show()
```

# Project 14 : OLED display aansturen met I2C

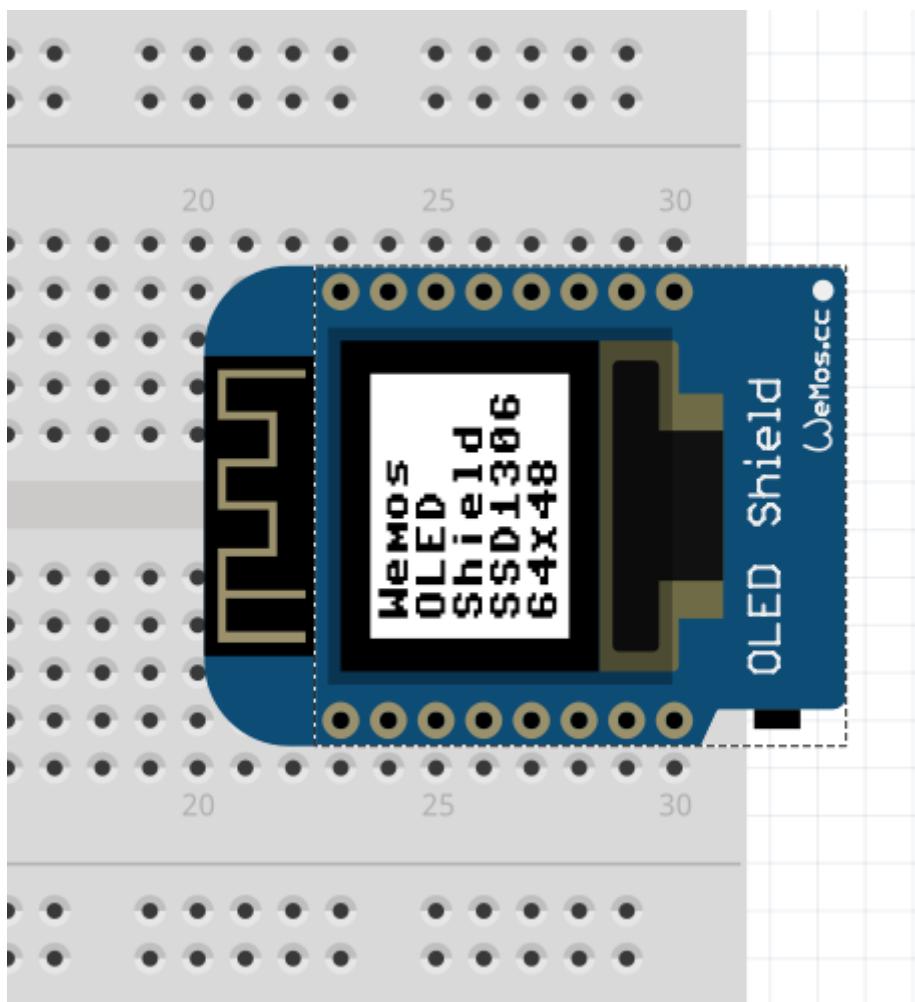
## Opgave

Toon de tekst “Hello World” op een OLED display dat aangestuurd wordt via I2C

## Materiaal

- Wemos D1 Mini bordje
- Wemos D1 Mini OLED shield

## Schakeling

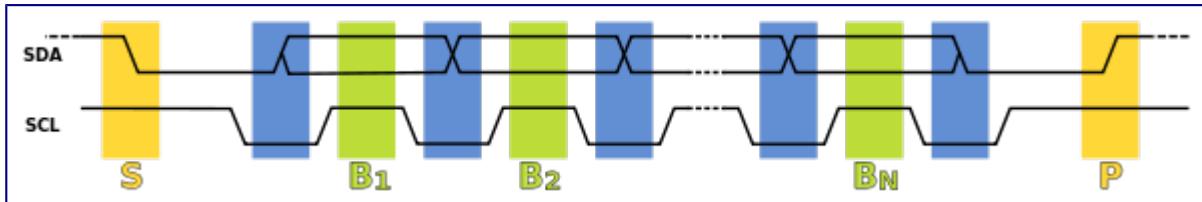


## I2C

I2C, vroeger aangeduid met IIC (Inter-IC) is een synchrone seriële bus voor datacommunicatie tussen microprocessoren en andere IC's. Het werd in 1979 door Philips ontwikkeld als goedkoop alternatief voor een reeks andere databussen. Soms spreekt men ook over TWI (Two Wire Interface).

I2C werkt op basis van twee bus lijnen, namelijk SDA (Serial DAta) en SCL (Serial CLock). Over SDA wordt de data verzonden en over SCL wordt het kloksignaal verzonden.

In het onderstaande timing diagram wordt verduidelijkt hoe de SDA en SCL samenwerken:



De werking van I2C dataoverdracht:

1. Data overdracht wordt geïnitieerd met een START bit (S) die SDA het signaal geeft om laag te gaan, terwijl SCL hoog blijft.
2. SDA zet de eerste databit klaar, terwijl SCL laag is (blauwe tijdsbalk).
3. SDA verstuur de data als SCL hoog gaat (groene tijdsbalk).
4. Het proces wordt herhaald voor alle volgende bits : klaarzetten bit bij lage SCL, versturen bij hoge SCL.
5. Data overdracht wordt gestopt met een STOP bit (P) die SDA hoog zet terwijl SCL continu hoog gehouden wordt.

Ten einde valse detecties te vermijden, wordt het klaarzetten van de bit door de SDA gedaan op de dalende flank (overgang hoog naar laag) van SCL. Het doorsturen gebeurt op de stijgende flank (de overgang laag naar hoog) van SCL.

Om te kunnen communiceren heeft I2C één master nodig en minimaal één slave. De master heeft de controle over de I2C-bus en genereert het kloksignaal, startbit en stopbit. De slaves communiceren alleen dan, nadat de master daartoe een verzoek stuurt.

Iedere slave hangt aan dezelfde I2C bus, maar heeft zijn eigen adres. De slave luistert enkel naar communicatie met zijn adres.

## Code

```
import ssd1306
from machine import I2C, Pin

# OLED
OLED_WIDTH  = 64
OLED_HEIGHT = 48

# I2C initialiseren
scl = Pin(5)                      # pin D1
sda = Pin(4)                      # pin D2
i2c = I2C(-1, scl, sda)          # -1 = software I2C

# OLED initialiseren
oled = ssd1306.SSD1306_I2C(OLED_WIDTH, OLED_HEIGHT, i2c)

# tekst plaatsen
oled.fill(0)
oled.text("Hello", 0, 0)
oled.text("World", 0, 10)
oled.pixel(20, 20, 1)
oled.show()
```

# Project 15 : LCD karakter display parallel aansturen

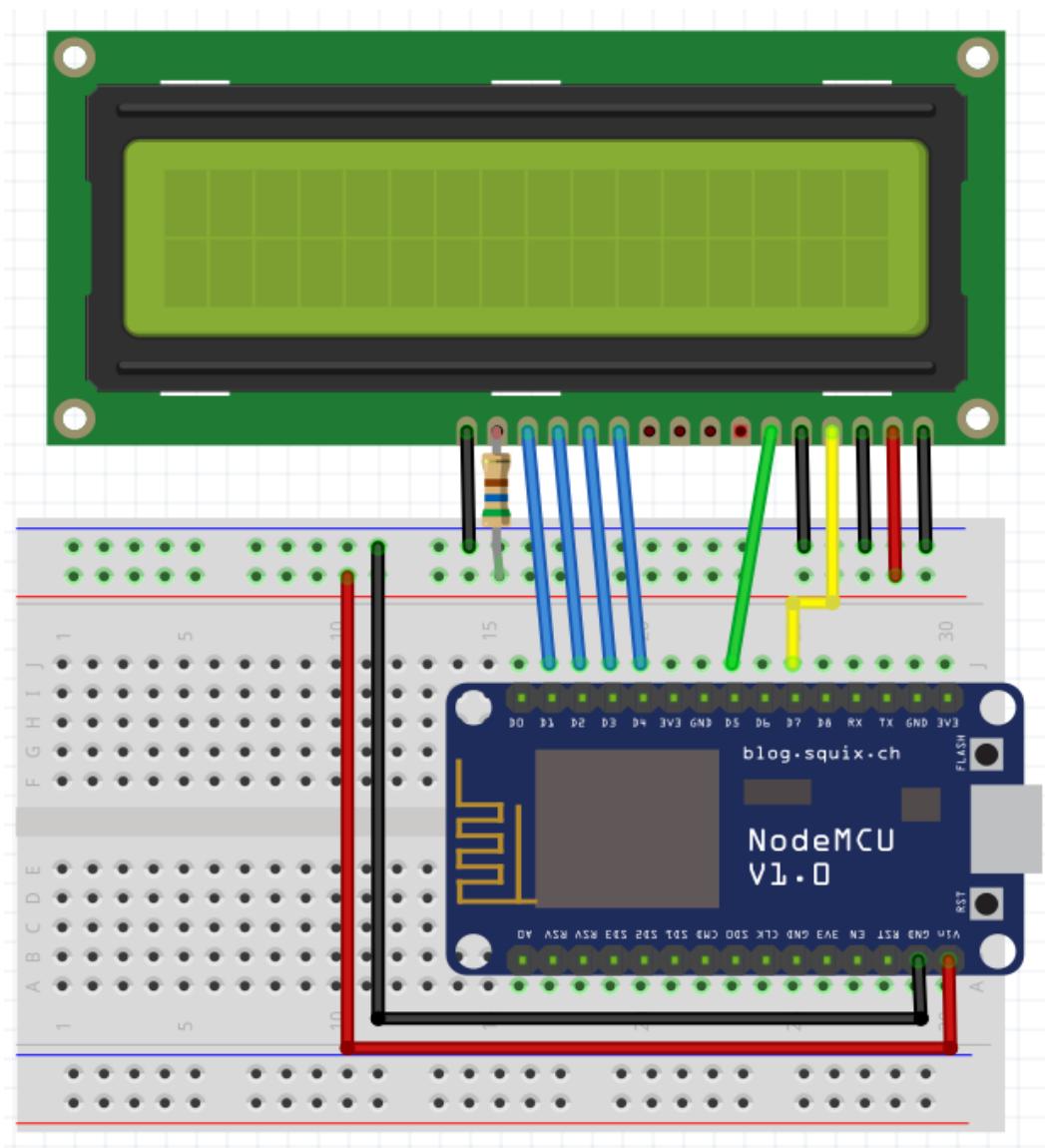
## Opgave

Toon wat tekst op een LCD karakter display dat parallel wordt aangestuurd.

## Materiaal

- NodeMCU ESP8266 bordje
- LCM1602C dot matrix karakter display (2 x 16 karakters)
- weerstand 560 ohm
- jumperkabels

## Schakeling

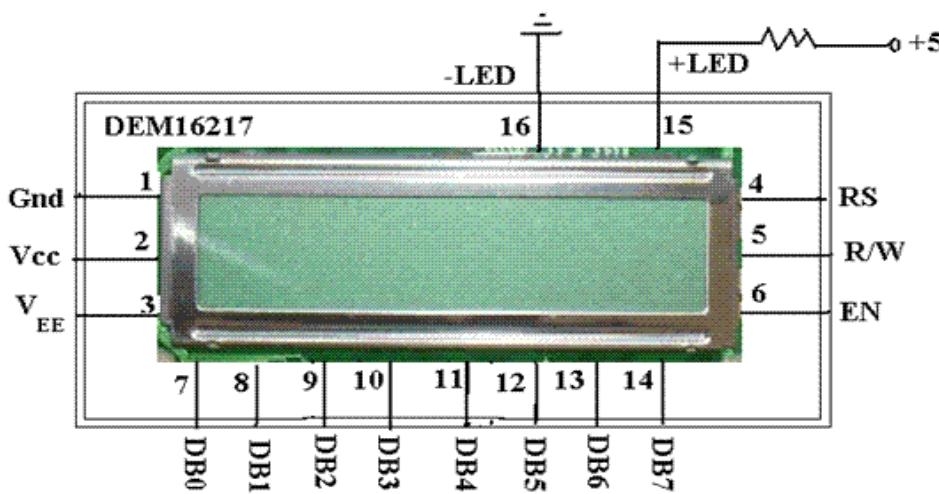


LCM1602 Pin	ESP8266 Pin	ESP8266 GPIO
GND	GND	-
VDD	VIN	-
VO	GND	-
RS	D7	13
RW	GND	-
ES	D5	14
DB0	-	*
DB1	-	-
DB2	-	-
DB3	-	-
DB4	D4	2
DB5	D3	0
DB6	D2	4
DB7	D1	5
BL1	VIN (via weerstand)	-
BL2	GND	-

## Dot matrix LCD

Er bestaan een hele reeks LCD dot matrix displays die bijna allemaal aangestuurd worden met een Hitachi HD44780 controller. De meeste van deze displays hebben een resolutie van 128x16 (2 lijnen), 128x32 (4 lijnen) of 128x64 (8 lijnen). De karakters zijn meestal opgebouwd als een 5x7 pixel matrix, of soms als 6x8. Deze displays hebben 14 tot 16 aansluit pinnen, waarvan 8 pinnen gebruikt worden om de bits van commando's en data door te sturen.

Pin configuration of LCD



## Code

*hd44780.py*

```

# Eenvoudige klasse voor een HD44780 gebaseerde LCD in 4-bit mode
# Deze displays hebben een 8-bit databus maar kunnen ook in 4-bit mode
# werken (de 4 lage bits DB0-DB3 worden dan niet gebruikt)
# Aansluitingen LCD :
# 1 : GND
# 2 : VDD (5V)
# 3 : VO (Contrast 0-5V)
# 4 : RS (Register select : low = Command - high = Data)
# 5 : RW (Read/Write : low = Write - high = Read)
# 6 : ES (Enable of Strobe)
# 7 : DB0 (Data Bit 0 - niet gebruikt in 4-bit mode)
# 8 : DB1 (Data Bit 1 - niet gebruikt in 4-bit mode)
# 9 : DB2 (Data Bit 2 - niet gebruikt in 4-bit mode)
# 10: DB3 (Data Bit 3 - niet gebruikt in 4-bit mode)
# 11: DB4 (Data Bit 4)
# 12: DB5 (Data Bit 5)
# 13: DB6 (Data Bit 6)
# 14: DB7 (Data Bit 7)
# 15: BL1 (Backlight 5V)
# 16: BL2 (Backlight GND)

import machine
import time

# gebruikte pinnen
PIN_RS = None
PIN_ES = None
PIN_D4 = None
PIN_D5 = None
PIN_D6 = None
PIN_D7 = None

pins = [PIN_RS, PIN_ES, PIN_D4, PIN_D5, PIN_D6, PIN_D7]

# dimensies LCD
LCD_COLS = 16    # LCM1602 = 16 - LCM2004 = 20
LCD_ROWS = 2      # LCM1602 = 2 - LCM2004 = 4

# LCD Register select
LCD_CHR = True
LCD_CMD = False

# LCD timings
E_PULSE = 0.005
E_DELAY = 0.001

class HD44780_LCD:

    def __init__(self, pins=[13, 14, 2, 0, 4, 5],
                 cols=LCD_COLS, rows=LCD_ROWS):
        self.pins = pins
        self.pin_rs = machine.Pin(self.pins[0], machine.Pin.OUT)
        self.pin_es = machine.Pin(self.pins[1], machine.Pin.OUT)
        self.pin_d4 = machine.Pin(self.pins[2], machine.Pin.OUT)
        self.pin_d5 = machine.Pin(self.pins[3], machine.Pin.OUT)

```

```

        self.pin_d6 = machine.Pin(self.pins[4], machine.Pin.OUT)
        self.pin_d7 = machine.Pin(self.pins[5], machine.Pin.OUT)
        self.cols = cols
        self.rows = rows

    def lcd_init(self):
        self.lcd_byte(0x33, LCD_CMD)      # 110011 Initialise
        self.lcd_byte(0x32, LCD_CMD)      # 110010 Initialise
        self.lcd_byte(0x0C, LCD_CMD)      # 000110 Cursor move direction
        self.lcd_byte(0x01, LCD_CMD)      # 001100 Display On, Cursor Off, Blink
Off
        self.lcd_byte(0x28, LCD_CMD)      # 101000 Data length, number of lines,
font size
        self.lcd_byte(0x06, LCD_CMD)      # 000001 Clear display
        time.sleep(E_DELAY)

    def lcd_byte(self, bits, mode):
        self.bits = bits
        self.mode = mode
        # Register select
        self.pin_rs.value(self.mode)
        # High bits
        self.pin_d4.low()
        self.pin_d5.low()
        self.pin_d6.low()
        self.pin_d7.low()
        if self.bits & 0x10 == 0x10:
            self.pin_d4.high()
        if self.bits & 0x20 == 0x20:
            self.pin_d5.high()
        if self.bits & 0x40 == 0x40:
            self.pin_d6.high()
        if self.bits & 0x80 == 0x80:
            self.pin_d7.high()
        # Toggle Enable
        self.lcd_toggle_enable()
        # Low bits
        self.pin_d4.low()
        self.pin_d5.low()
        self.pin_d6.low()
        self.pin_d7.low()
        if self.bits & 0x01 == 0x01:
            self.pin_d4.high()
        if self.bits & 0x02 == 0x02:
            self.pin_d5.high()
        if self.bits & 0x04 == 0x04:
            self.pin_d6.high()
        if self.bits & 0x08 == 0x08:
            self.pin_d7.high()
        # Toggle Enable
        self.lcd_toggle_enable()

    def lcd_toggle_enable(self):
        time.sleep(E_DELAY)
        self.pin_es.high()
        time.sleep(E_PULSE)
        self.pin_es.low()
        time.sleep(E_DELAY)

```

```

def lcd_string(self, string, row=1, col=0):
    self.string = string
    # controle positie
    if row > self.rows:
        row = self.rows
    if col > self.cols:
        col = self.cols
    # adres in buffer
    if row == 1:
        self.lcd_byte(0x80 + col, LCD_CMD)
    elif row == 2:
        self.lcd_byte(0xC0 + col, LCD_CMD)
    elif row == 3:
        self.lcd_byte(0x94 + col, LCD_CMD)
    elif row == 4:
        self.lcd_byte(0xD4 + col, LCD_CMD)
    # Druk string af
    for i in range(len(self.string)):
        if i <= self.cols:
            self.lcd_byte(ord(self.string[i]), LCD_CHR)
    # Vul rest van lijn met spaties
    for i in range(self.cols - len(self.string)):
        self.lcd_byte(32, LCD_CHR)

```

### ***testlcd.py***

```

import hd44780

# Initialisatie LCM1602C 16x2 karakter display
# Pins: RS - ES - DB4 - DB5 - DB6 - DB7
# GPIO: 13 - 14 - 2 - 0 - 4 - 5
pins = [13, 14, 2, 0, 4, 5]
lcd = hd44780.HD44780_LCD(pins, 16, 2)

# lcd initialiseren
lcd.lcd_init()

# tekst op display zetten
lcd.lcd_string("HD44780 LCD", 1, 0)
lcd.lcd_string("met MicroPython", 2, 0)

```

# Project 16 : LCD karakter display aansturen met I2C

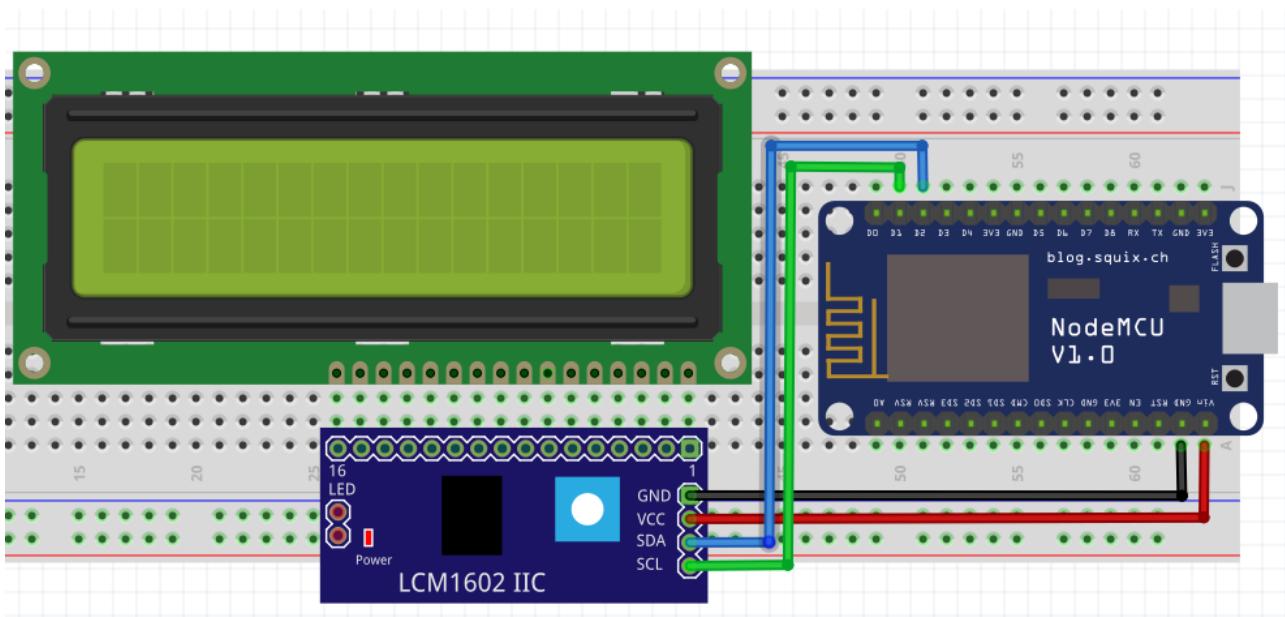
## Opgave

Toon wat tekst op een LCD karakter display dat via I2C wordt aangestuurd.

## Materiaal

- NodeMCU ESP8266 bordje
- LCM1602C dot matrix karakter display (2 x 16 karakters)
- PCF8574 I2C backpack module voor LCD karakter displays
- jumperkabels

## Schakeling



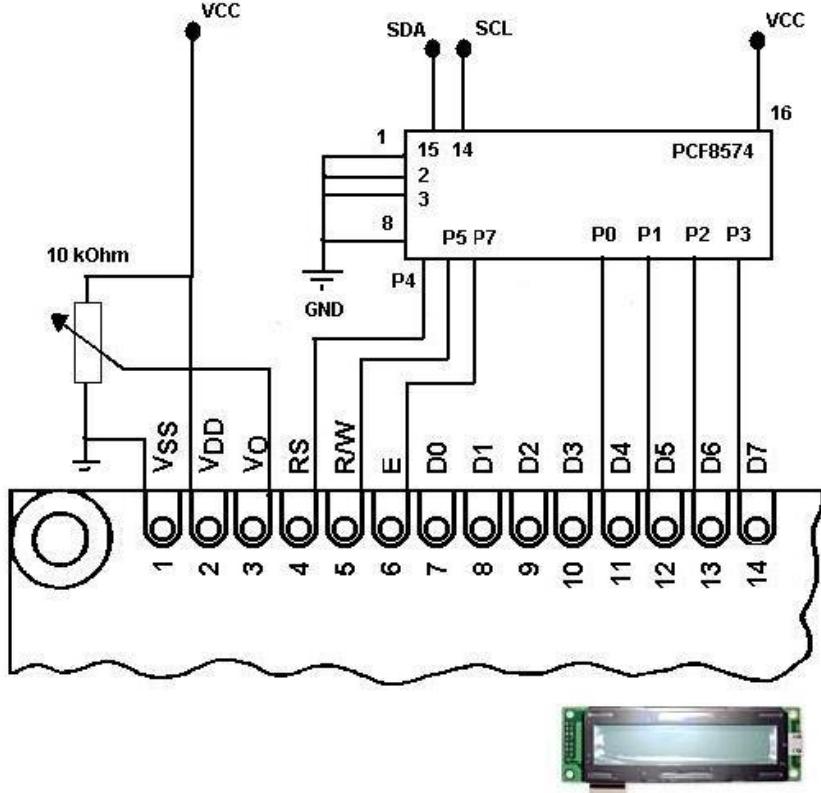
LCM1602 I2C Pin	ESP8266 Pin	ESP8266 GPIO
GND	GND	-
VCC	VIN	-
SDA	D2	4
SCL	D1	5

## I2C backpack module

Er bestaan een hele reeks LCD dot matrix displays die bijna allemaal aangestuurd worden met een Hitachi HD44780 controller. Deze displays hebben 14 tot 16 aansluit pinnen, wat uiteraard niet praktisch is voor onze kleine microcontrollers met hun beperkt aantal I/O aansluitingen.

Daarom zijn de meeste van deze displays voorzien van een PCF8574 I2C backpack module zodat het aantal aansluitingen gereduceerd wordt tot 4 (VCC, GND, SCL (klok) en SDA (data))

Connection of the PCF8574 to a LCD



## Code

### Lcd\_api.py

```
"""Provides an API for talking to HD44780 compatible character LCDs."""
import time

class LcdApi:
    """Implements the API for talking with HD44780 compatible character LCDs.
    This class only knows what commands to send to the LCD, and not how to get
    them to the LCD.

    It is expected that a derived class will implement the hal_XXX functions.
    """

    # The following constant names were lifted from the avrlib lcd.h
    # header file, however, I changed the definitions from bit numbers
    # to bit masks.
    #
    # HD44780 LCD controller command set

    LCD_CLR = 0x01          # DB0: clear display
    LCD_HOME = 0x02          # DB1: return to home position

    LCD_ENTRY_MODE = 0x04    # DB2: set entry mode
    LCD_ENTRY_INC = 0x02     # --DB1: increment
    LCD_ENTRY_SHIFT = 0x01   # --DB0: shift
```

```

LCD_ON_CTRL = 0x08          # DB3: turn lcd/cursor on
LCD_ON_DISPLAY = 0x04        # --DB2: turn display on
LCD_ON_CURSOR = 0x02         # --DB1: turn cursor on
LCD_ON_BLINK = 0x01          # --DB0: blinking cursor

LCD_MOVE = 0x10              # DB4: move cursor/display
LCD_MOVE_DISP = 0x08          # --DB3: move display (0-> move cursor)
LCD_MOVE_RIGHT = 0x04         # --DB2: move right (0-> left)

LCD_FUNCTION = 0x20           # DB5: function set
LCD_FUNCTION_8BIT = 0x10      # --DB4: set 8BIT mode (0->4BIT mode)
LCD_FUNCTION_2LINES = 0x08    # --DB3: two lines (0->one line)
LCD_FUNCTION_10DOTS = 0x04    # --DB2: 5x10 font (0->5x7 font)
LCD_FUNCTION_RESET = 0x30     # See "Initializing by Instruction" section

LCD_CGRAM = 0x40              # DB6: set CG RAM address
LCD_DDRAM = 0x80              # DB7: set DD RAM address

LCD_RS_CMD = 0
LCD_RS_DATA = 1

LCD_RW_WRITE = 0
LCD_RW_READ = 1

def __init__(self, num_lines, num_columns):
    self.num_lines = num_lines
    if self.num_lines > 4:
        self.num_lines = 4
    self.num_columns = num_columns
    if self.num_columns > 40:
        self.num_columns = 40
    self.cursor_x = 0
    self.cursor_y = 0
    self.backlight = True
    self.display_off()
    self.backlight_on()
    self.clear()
    self.hal_write_command(self.LCD_ENTRY_MODE | self.LCD_ENTRY_INC)
    self.hide_cursor()
    self.display_on()

def clear(self):
    """Clears the LCD display and moves the cursor to the top left corner."""
    self.hal_write_command(self.LCD_CLR)
    self.hal_write_command(self.LCD_HOME)
    self.cursor_x = 0
    self.cursor_y = 0

def show_cursor(self):
    """Causes the cursor to be made visible."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                           self.LCD_ON_CURSOR)

def hide_cursor(self):
    """Causes the cursor to be hidden."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

```

```

def blink_cursor_on(self):
    """Turns on the cursor, and makes it blink."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                           self.LCD_ON_CURSOR | self.LCD_ON_BLINK)

def blink_cursor_off(self):
    """Turns on the cursor, and makes it no blink (i.e. be solid)."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                           self.LCD_ON_CURSOR)

def display_on(self):
    """Turns on (i.e. unblanks) the LCD."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

def display_off(self):
    """Turns off (i.e. blanks) the LCD."""
    self.hal_write_command(self.LCD_ON_CTRL)

def backlight_on(self):
    """Turns the backlight on.

    This isn't really an LCD command, but some modules have backlight
    controls, so this allows the hal to pass through the command.
    """
    self.backlight = True
    self.hal_backlight_on()

def backlight_off(self):
    """Turns the backlight off.

    This isn't really an LCD command, but some modules have backlight
    controls, so this allows the hal to pass through the command.
    """
    self.backlight = False
    self.hal_backlight_off()

def move_to(self, cursor_x, cursor_y):
    """Moves the cursor position to the indicated position. The cursor
    position is zero based (i.e. cursor_x == 0 indicates first column).
    """
    self.cursor_x = cursor_x
    self.cursor_y = cursor_y
    addr = cursor_x & 0x3f
    if cursor_y & 1:
        addr += 0x40      # Lines 1 & 3 add 0x40
    if cursor_y & 2:
        addr += 0x14      # Lines 2 & 3 add 0x14
    self.hal_write_command(self.LCD_DDRAM | addr)

def putchar(self, char):
    """Writes the indicated character to the LCD at the current cursor
    position, and advances the cursor by one position.
    """
    if char != '\n':
        self.hal_write_data(ord(char))
        self.cursor_x += 1
    if self.cursor_x >= self.num_columns or char == '\n':
        self.cursor_x = 0
        self.cursor_y += 1
        if self.cursor_y >= self.num_lines:
            self.cursor_y = 0
        self.move_to(self.cursor_x, self.cursor_y)

```

```

def putstr(self, string):
    """Write the indicated string to the LCD at the current cursor
    position and advances the cursor position appropriately.
    """
    for char in string:
        self.putchar(char)

def custom_char(self, location, charmap):
    """Write a character to one of the 8 CGRAM locations, available
    as chr(0) through chr(7).
    """
    location &= 0x7
    self.hal_write_command(self.LCD_CGRAM | (location << 3))
    time.sleep_us(40)
    for i in range(8):
        self.hal_write_data(charmap[i])
        time.sleep_us(40)
    self.move_to(self.cursor_x, self.cursor_y)

def hal_backlight_on(self):
    """Allows the hal layer to turn the backlight on.

    If desired, a derived HAL class will implement this function.
    """
    pass

def hal_backlight_off(self):
    """Allows the hal layer to turn the backlight off.

    If desired, a derived HAL class will implement this function.
    """
    pass

def hal_write_command(self, cmd):
    """Write a command to the LCD.

    It is expected that a derived HAL class will implement this
    function.
    """
    raise NotImplementedError

def hal_write_data(self, data):
    """Write data to the LCD.

    It is expected that a derived HAL class will implement this
    function.
    """
    raise NotImplementedError

```

## *esp8266\_i2c\_lcd.py*

```
"""Implements a HD44780 character LCD connected via PCF8574 on I2C.  
This was tested with: https://www.wemos.cc/product/d1-mini.html"""\n\nfrom lcd_api import LcdApi  
from time import sleep_ms\n\n# The PCF8574 has a jumper selectable address: 0x20 - 0x27  
DEFAULT_I2C_ADDR = 0x27\n\n# Defines shifts or masks for the various LCD line attached to the PCF8574\n\nMASK_RS = 0x01  
MASK_RW = 0x02  
MASK_E = 0x04  
SHIFT_BACKLIGHT = 3  
SHIFT_DATA = 4\n\n\nclass I2cLcd(LcdApi):\n    """Implements a HD44780 character LCD connected via PCF8574 on I2C."""\n\n    def __init__(self, i2c, i2c_addr, num_lines, num_columns):\n        self.i2c = i2c\n        self.i2c_addr = i2c_addr\n        self.i2c.writeto(self.i2c_addr, bytarray([0]))\n        sleep_ms(20)    # Allow LCD time to powerup\n        # Send reset 3 times\n        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)\n        sleep_ms(5)    # need to delay at least 4.1 msec\n        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)\n        sleep_ms(1)\n        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)\n        sleep_ms(1)\n        # Put LCD into 4 bit mode\n        self.hal_write_init_nibble(self.LCD_FUNCTION)\n        sleep_ms(1)\n        LcdApi.__init__(self, num_lines, num_columns)\n        cmd = self.LCD_FUNCTION\n        if num_lines > 1:\n            cmd |= self.LCD_FUNCTION_2LINES\n        self.hal_write_command(cmd)\n\n    def hal_write_init_nibble(self, nibble):\n        """Writes an initialization nibble to the LCD.\n\n        This particular function is only used during initialization.\n        """\n        byte = ((nibble >> 4) & 0x0f) << SHIFT_DATA\n        self.i2c.writeto(self.i2c_addr, bytarray([byte | MASK_E]))\n        self.i2c.writeto(self.i2c_addr, bytarray([byte]))\n\n    def hal_backlight_on(self):\n        """Allows the hal layer to turn the backlight on."""\n        self.i2c.writeto(self.i2c_addr, bytarray([1 << SHIFT_BACKLIGHT]))
```

```

def hal_backlight_off(self):
    """Allows the hal layer to turn the backlight off."""
    self.i2c.writeto(self.i2c_addr, bytearray([0]))

def hal_write_command(self, cmd):
    """Writes a command to the LCD.

    Data is latched on the falling edge of E.
    """
    byte = ((self.backlight << SHIFT_BACKLIGHT) |
            (((cmd >> 4) & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytearray([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytearray([byte]))
    byte = ((self.backlight << SHIFT_BACKLIGHT) |
            ((cmd & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytearray([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytearray([byte]))
    if cmd <= 3:
        # home and clear commands require a worst case delay of 4.1 msec
        sleep_ms(5)

def hal_write_data(self, data):
    """Write data to the LCD."""
    byte = (MASK_RS | (self.backlight << SHIFT_BACKLIGHT) |
            (((data >> 4) & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytearray([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytearray([byte]))
    byte = (MASK_RS | (self.backlight << SHIFT_BACKLIGHT) |
            ((data & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytearray([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytearray([byte]))

```

### ***esp8266\_i2c\_lcd\_test.py***

```

"""Implements a HD44780 character LCD connected via PCF8574 on I2C.
This was tested with: https://www.wemos.cc/product/d1-mini.html"""

from time import sleep_ms, ticks_ms
from machine import I2C, Pin
from esp8266_i2c_lcd import I2cLcd

# The PCF8574 has a jumper selectable address: 0x20 - 0x27
DEFAULT_I2C_ADDR = 0x27

def test_main():
    """Test function for verifying basic functionality."""
    print("Running test_main")
    i2c = I2C(scl=Pin(5), sda=Pin(4), freq=400000)
    lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
    lcd.putstr("It Works!\nSecond Line")
    sleep_ms(3000)
    lcd.clear()
    count = 0
    while True:
        lcd.move_to(0, 0)

```

```
lcd.putstr("%7d" % (ticks_ms() // 1000))
sleep_ms(1000)
count += 1
if count % 10 == 3:
    print("Turning backlight off")
    lcd.backlight_off()
if count % 10 == 4:Schermafdruk van 2017-04-10 14-39-02
    print("Turning backlight on")
    lcd.backlight_on()
if count % 10 == 5:
    print("Turning display off")
    lcd.display_off()
if count % 10 == 6:
    print("Turning display on")
    lcd.display_on()
if count % 10 == 7:
    print("Turning display & backlight off")
    lcd.backlight_off()
    lcd.display_off()
if count % 10 == 8:
    print("Turning display & backlight on")
    lcd.backlight_on()
    lcd.display_on()

# if __name__ == "__main__":
test_main()
```

# Project 17 : NeoPixel matrix aansturen

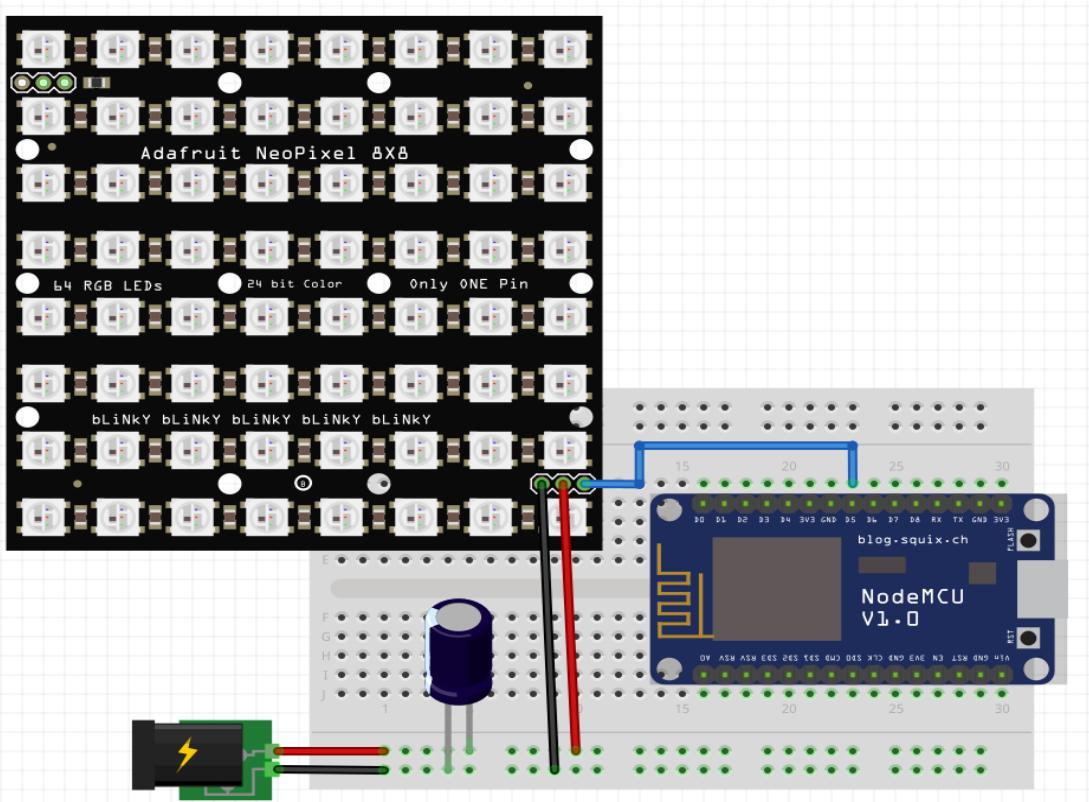
## Opgave

Stuur een matrix met NeoPixels aan met verschillende kleuren en effecten.

## Materiaal

- NodeMCU ESP8266 bordje
- CJMCU-8\*8 NeoPixel matrix
- externe 5V voeding + 470 µF elco
- jumperkabels

## Schakeling



## NeoPixel

NeoPixels zijn adresseerbare RGB leds van Adafruit. Het bestaat uit een kleine surface mounted package waarin een rode, groene en blauwe led samen met een controller chip gemonteerd is. Het geheel wordt aangestuurd met 1 pin.

NeoPixel strips en matrices zijn gebaseerd op de WS2812, WS2811 en SK6812 leds/drivers en gebruiken een eendraadprotocol.



## Code

```
from machine import Pin
import neopixel
import time

# constanten
PIXEL_PIN = Pin(14, Pin.OUT)                      # Pin D5
PIXEL_AANTAL = 64                                  # 8 * 8 neopixels

def demo(np):
    n = np.n

    # cycle
    for i in range(4 * n):
        for j in range(n):
            np[j] = (0, 0, 0)
        np[i % n] = (255, 255, 255)
        np.write()
        time.sleep_ms(25)

    # bounce
    for i in range(4 * n):
        for j in range(n):
            np[j] = (0, 0, 128)
        if (i // n) % 2 == 0:
            np[i % n] = (0, 0, 0)
        else:
            np[n - 1 - (i % n)] = (0, 0, 0)
        np.write()
        time.sleep_ms(60)

    # fade in/out
    for i in range(0, 4 * 256, 8):
        for j in range(n):
            if (i // 256) % 2 == 0:
                val = i & 0xff
            else:
                val = 255 - (i & 0xff)
            np[j] = (val, 0, 0)
        np.write()

    # clear
    for i in range(n):
        np[i] = (0, 0, 0)
    np.write()

# initialisatie neopixel matrix
np = neopixel.NeoPixel(PIXEL_PIN, PIXEL_AANTAL)

# demo neopixels
demo(np)
```

# Project 18 : ESP8266 als WiFi client

## Opgave

Maak een klasse om de ESP8266 te verbinden met een lokaal netwerk, een socket server en een socket client. Test de wifi functionaliteit van de klasse uit in de seriële REPL.

## Materiaal

- NodeMCU ESP8266 bordje

## Code

### *netconfig.py*

```
class config:  
    #settings for wireless connection and socket connection  
    ssid_to_cn="KruisA"  
    passwd="Hetcvo.be"  
    static_ip=""  
    connect_to="192.168.0.108"  
    connect_to_port=8100  
    port=1022  
  
    #vars for wlan, socket server and client  
    wlan = None  
    sSocket = None  
    cSocket = None  
    conn=None  
    NUMBYTES=const(32)
```

## ***networking.py***

```
import netconfig as net
import network
import socket

class wifi:
    @staticmethod
    def connectWIFI():
        net.config.wlan = network.WLAN(network.STA_IF)
        net.config.wlan.active(True)
        if not net.config.wlan.isconnected():
            print('connecting to network...')
            net.config.wlan.connect(net.config.ssid_to_cn, net.config.passwd)
            while not net.config.wlan.isconnected():
                pass
        #if in net.config static_ip is defined change ip adress obtained by
DHCP with the static adress
        if net.config.static_ip != "":
            ifData=list(net.config.wlan.ifconfig())
            ifData[0]=net.config.static_ip
            net.config.wlan.ifconfig(tuple(ifData))
        print('network config:', net.config.wlan.ifconfig())
        return net.config.wlan

    @staticmethod
    def disconnectWIFI():
        if net.config.wlan == None:
            return 0
        if not net.config.wlan.isconnected():
            return 0
        net.config.wlan.disconnect()
        return 1

class socketServer:

    @staticmethod
    def start():
        try:
            host = net.config.wlan.ifconfig()[0]
            net.config.sSocket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

            net.config.sSocket.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
            addr = socket.getaddrinfo(host,net.config.port)
            print(addr)
            net.config.sSocket.bind(addr[0][-1])
            net.config.sSocket.listen(1)
            return True
        except Exception as e:
            net.config.sSocket.close()
            print(e)
            return False

    @staticmethod
    def waitCn():
        try:
            net.config.conn,addr=net.config.sSocket.accept()
            qs = net.config.conn.recv(net.config.NUMBYTES).decode("ascii")
```

```

        if qs == 'GO' or qs == 'go':
            return 1
        elif qs == 'HALT' or qs == 'halt':
            net.config.conn.close()
            net.config.sSocket.close()
            return 0
        else:
            net.config.conn.close()
            net.config.conn=None
            return -1
    except Exception as e:
        print(e)
        net.config.conn.close()
        return -1

    @staticmethod
    def sendData(data):
        net.config.conn.send(data)

    @staticmethod
    def readData():
        res=net.config.conn.recv(net.config.NUMBYTES)
        return res.decode("ascii")
    @staticmethod
    def stopClient():
        try:
            net.config.conn.close()
            return True
        except:
            return True

    @staticmethod
    def stop():
        try:
            net.config.sSocket.close()
            return True
        except:
            return True

class socketHTTPRequest:

    @staticmethod
    def sendGETRequest(req):
        try:
            addr =
socket.getaddrinfo(net.config.connect_to,net.config.connect_to_port)[0][-1]
            cSocket=socket.socket()
            cSocket.connect(addr)
            strToSend='GET ' + req + ' HTTP/1.1\r\n\r\n'
            cSocket.send(strToSend.encode("ascii"))
            res=cSocket.recv(1024)
            res=res.decode("ascii")
            if res.find("OK")<0 or res.find("200")<0:
                cSocket.close()
                return 0
            cSocket.close()
            return 1
        except Exception as e:
            print(e)
            return -1

```

## ***Uittesten in REPL***

```
MicroPython v1.8.7-7-gb5a1a20a3 on 2017-01-09; ESP module with ESP8266
Type "help()" for more information.
>>> import networking
>>> networking.wifi.connectWIFI()
network config: ('192.168.64.20', '255.255.255.0', '192.168.1.1',
'192.168.1.2')
<WLAN>
>>> networking.wifi.disconnectWIFI()
1
>>>
```

# Project 19 : ESP8266 als Socket Server

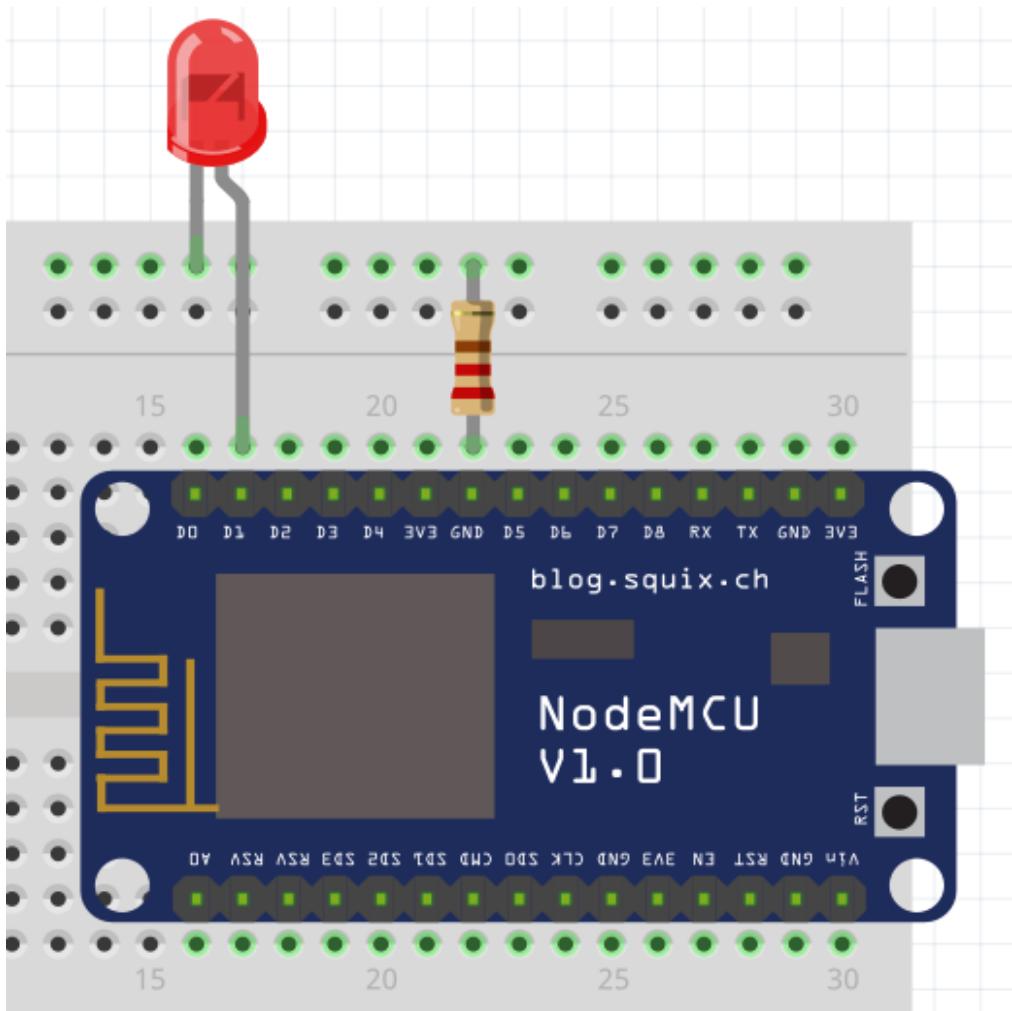
## Opgave

Test de socket server functionaliteit van de vorige klasse uit. Voorzie je ESP van een LED en schrijf een socket server programma. Op de Raspberry Pi schrijven we een socket client programma die via de communicatie over het lokale netwerk de LED aan en uit zet en tevens de socket server op de ESP kan stoppen. Als bonus maken we een webapplicatie op de Pi die de LED aan en uit kan zetten en de socketserver op de ESP kan stoppen.

## Materiaal

- NodeMCU ESP8266 bordje
- LED
- weerstand 220 ohm
- jumperkabels

## Schakeling



## Code

### *testSocketServer.py*

```
from machine import Pin
from networking import wifi, socketServer

def app():
    GPIO0 = 5
    pinLED = Pin(GPIO0, Pin.OUT)
    pinLED.low()
    # connecteren met netwerk
    wifi.connectWIFI()
    if socketServer.start() == False:
        wifi.disconnectWIFI()
        return 0

    while True:
        status = socketServer.waitCn()
        if status < 0:
            print("Wrong command")
        elif status == 0:
            print("Halt ESP")
            break
        else:
            socketServer.sendData("OK")
            res = socketServer.readData()
            pinLED.value(int(res))
            socketServer.sendData("OK")
            socketServer.stopClient()

    pinLED.low()
    wifi.disconnectWIFI()

app()
```

### *testSocketCl.py*

```
from networkingPI import PiSocketClient
import time

IPSERVER = "10.97.161.216"
PORT = 1022
LEDseq = [1,1,1,1,0,0,1,0,1,0,1,0,1,1,1,1,0,0,1,1,1,1,0]

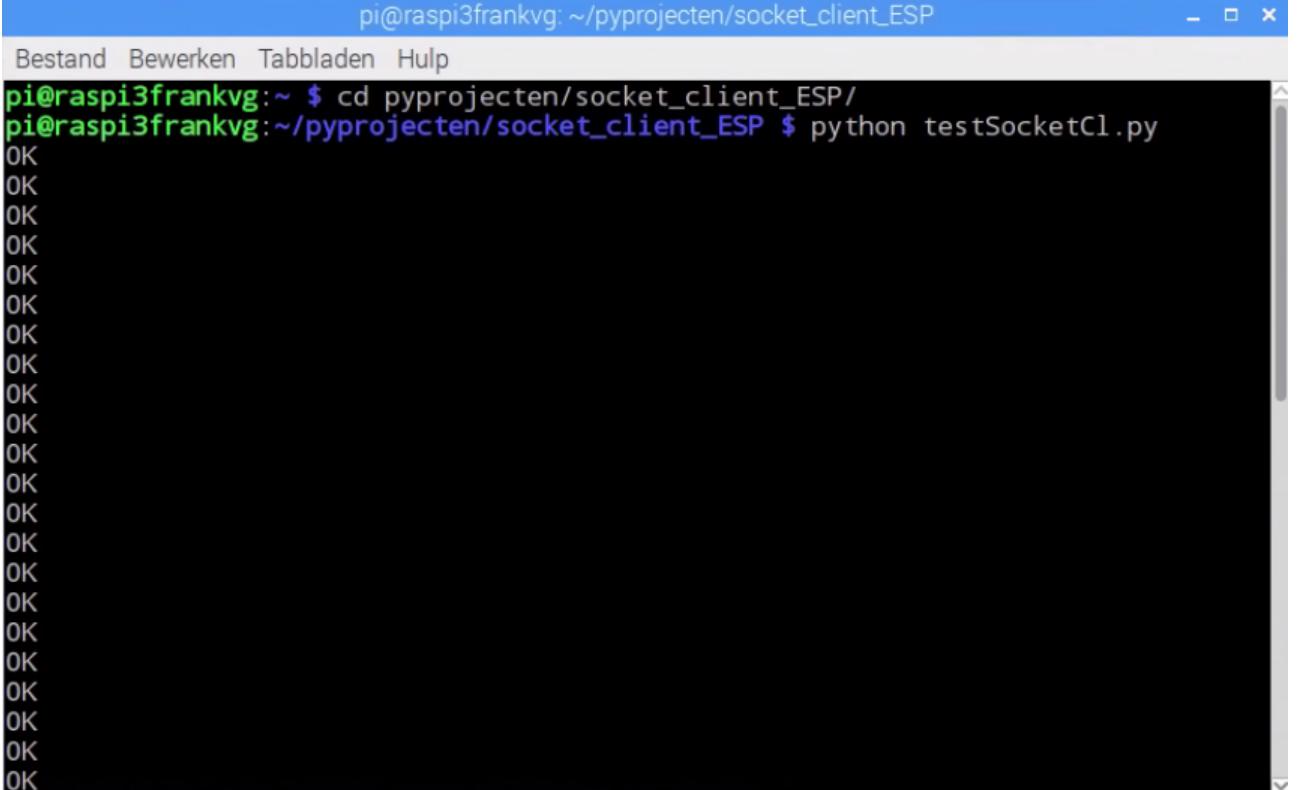
for s in LEDseq:
    if PiSocketClient.start(IPSERVER, PORT):
        PiSocketClient.sendData("go")
        res = PiSocketClient.readData()
        print(res)
        PiSocketClient.sendData(str(s))
        res = PiSocketClient.readData()
        print(res)
    PiSocketClient.stop()
    time.sleep(2)

PiSocketClient.start(IPSERVER, PORT)
PiSocketClient.sendData("halt")
PiSocketClient.stop()
```

### **ESP8266 REPL output**

```
>>> import testSocketServer
connecting to network...
network config: ('192.168.1.30', '255.255.255.0', '192.168.1.1',
'192.168.1.2')
[(2, 1, 0, '', ('192.168.1.30', 1022))]
Halt ESP
>>>
```

### **Raspberry Pi output**



A screenshot of a terminal window titled "pi@raspi3frankvg: ~/pyprojecten/socket\_client\_ESP". The window has standard OS X-style window controls at the top right. The menu bar includes "Bestand", "Bewerken", "Tabbladen", and "Hulp". The terminal content shows the command "cd pyprojecten/socket\_client\_ESP/" followed by "python testSocketCl.py". The output consists of 20 lines of "OK" text, indicating successful communication between the client and server.

```
pi@raspi3frankvg:~/pyprojecten/socket_client_ESP$ python testSocketCl.py
OK
```

## AppServers/Led1Esp/webpy/worker.py

```
import threading
import json
from networkingPI import PiSocketClient

class work:

    data={"button_3":"","toggle_2":"0","output_0":""}
    lock = threading.Lock()
    #schrijf hier eigen klas variabelen, zoals
    #Lists voor selects en radiogroups
    IPSEVER = "192.168.1.30"
    PORT = 1022

    @classmethod
    def start(cls):
        #schrijf hieronder je initialisatie code:
        #init gpio pinnen (RPi.GPIO), init sercomm ...
        print("start worker")

    @classmethod
    def stop(cls):
        #schrijf hieronder je opkuis code:
        #opkuisen gpio pinnen (RPi.GPIO), sluiten sercomm ...
        print("stop worker")

    @classmethod
    def do_btnStopESP(cls):
        try:
            cls.lock.acquire()
            #schrijf hier code, bijv. getters en/of setters
            PiSocketClient.start(cls.IPSEVER, cls.PORT)
            PiSocketClient.sendData("halt")
            PiSocketClient.stop()
        finally:
            cls.lock.release()

        return json.dumps(cls.data,ensure_ascii=False)

    @classmethod
    def get_btnStopESP(cls):
        return cls.data["button_3"]

    @classmethod
    #maak dat een setter altijd tussen cls.lock.acquire()
    #en cls.lock.release() staat!!!!
    def set_btnStopESP(cls,val):
        cls.data["button_3"]=val

    @classmethod
    def do_tgLedESP(cls,val):
        try:
            cls.lock.acquire()
            cls.set_tgLedESP(val)
            #schrijf hier code, bijv. aan de hand van val een LED doen
            branden
        finally:
            cls.lock.release()
```

```

        if PiSocketClient.start(cls.IPSERVER, cls.PORT):
            PiSocketClient.sendData("go")
            res = PiSocketClient.readData()
            PiSocketClient.sendData(val)
            res = PiSocketClient.readData()
    finally:
        cls.lock.release()

    return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_tgLedESP(cls):
    return cls.data["toggle_2"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_tgLedESP(cls,val):
    cls.data["toggle_2"]=val

@classmethod
def do_output_0(cls):
    try:
        cls.lock.acquire()
        #schrijf hier code, bijv. getters en/of setters
    finally:
        cls.lock.release()

    return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_output_0(cls):
    return cls.data["output_0"]

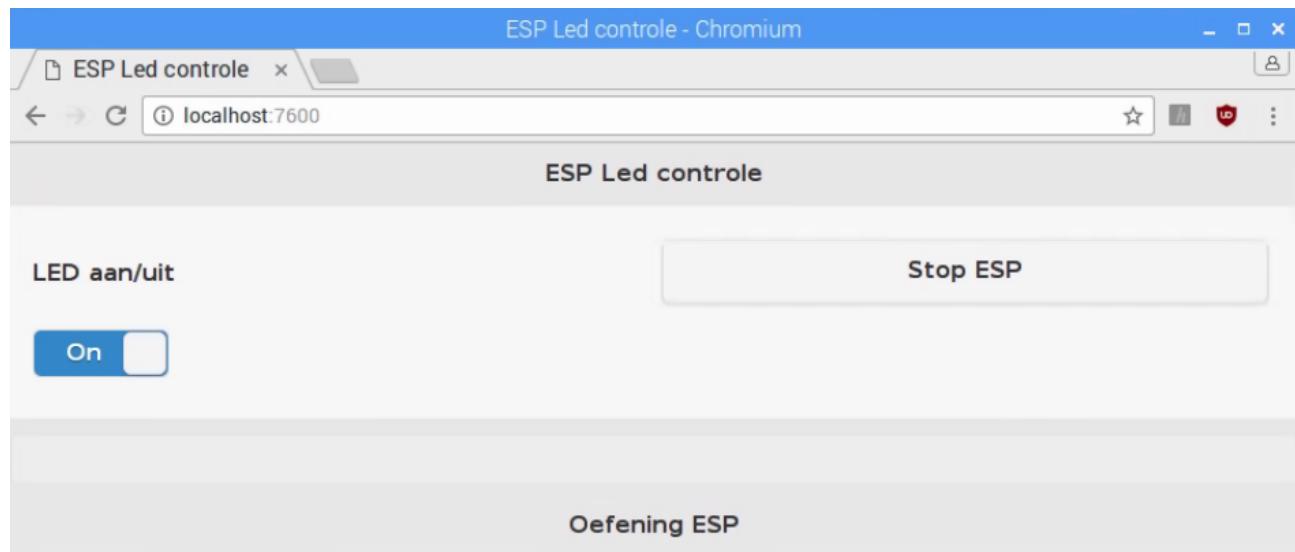
@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_output_0(cls,val):
    cls.data["output_0"]=val

```

## Webapplicatie console

```
pi@raspi3frankvg:~/AppServers/Led1Esp/webpy
Bestand Bewerken Tabbladen Hulp
pi@raspi3frankvg:~/AppServers/Led1Esp/webpy/
pi@raspi3frankvg:~/AppServers/Led1Esp/webpy $ python server.py 7600
start worker
http://0.0.0.0:7600/
127.0.0.1:57634 - - [15/Mar/2017 19:17:03] "HTTP/1.1 GET /" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57644 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /favicon.ico" - 404 Not Found
127.0.0.1:57648 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /toggle_2/1" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:34] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:36] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:38] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:40] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57638 - - [15/Mar/2017 19:17:44] "HTTP/1.1 GET /output_0" - 200 OK
127.0.0.1:57644 - - [15/Mar/2017 19:17:44] "HTTP/1.1 GET /toggle_2/0" - 200 OK
127.0.0.1:57644 - - [15/Mar/2017 19:17:44] "HTTP/1.1 GET /output_0" - 200 OK
```

## Webapplicatie GUI



# Project 20 : ESP8266 als Socket Client

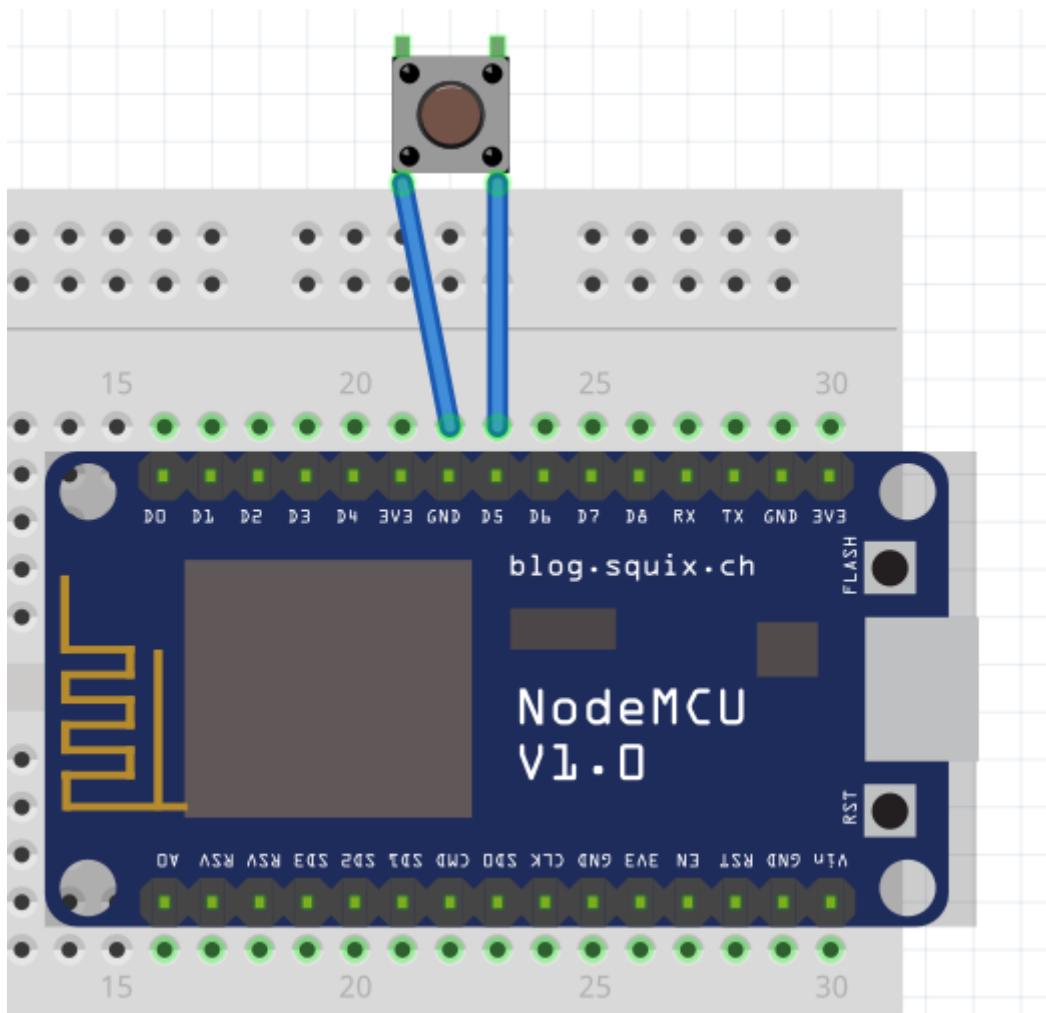
## Opgave

Test de socket client functionaliteit van de vorige klasse uit. Verbind een drukknop met de ESP8266 en toon in de webapplicatie op de Raspberry Pi het aantal keer dat de drukknop is ingedrukt. Gebruik hiervoor een button event om het drukken te detecteren en een output om het aantal drukken te tonen. De button zelf mag onzichtbaar zijn.

## Materiaal

- NodeMCU ESP8266 bordje
- jumperkabels

## Schakeling



We activeren de ingebouwde pullup weerstand op pin D5 (GPIO 14) in de code. Bij het indrukken van de knop wordt D5 dan laag en wordt tevens de stroom op de poort beperkt.

## Code

### *testESPbutton.py*

```
from machine import Pin
from networking import wifi, socketHTTPRequest
import time

# gpio button met pullup weerstand
ButNum = 14
button = Pin(ButNum, Pin.IN, Pin.PULL_UP)

# connecteren op wifi
wifi.connectWIFI()

while True:
    # knop gedrukt
    if not button.value():
        # stuur GET request naar button op webserver RPi
        res = socketHTTPRequest.sendGETRequest("/button_2")
        print("button pushed GET request: {}".format(res))
        # even wachten
        time.sleep_ms(300)
        # wachten op knop los
        while not button.value():
            pass

# disconnect van wifi
wifi.disconnectWIFI()
```

### *ESP8266 REPL output*

```
>>> import testESPbutton
network config: ('192.168.1.30', '255.255.255.0', '192.168.1.1',
'192.168.1.2')
button pushed GET request: 1
```

### *AppServers/testESPbutton/webpy/templates/index.html*

(onzichtbaar maken van button)

```
...
<div data-role="content">
<div class="ui-grid-a">
    <div class="ui-block-a"><span style="display:none"><input type="button"
value="Hidden" id="button_2"></span></div>
    <div class="ui-block-b"><h4>Teller</h4>
<div data-role="fieldcontain"><input type="text" id="output_3" />
</div></div>
</div>
...
```

## AppServers/testESPbutton/webpy/worker.py

```
import threading
import json

class work:

    data={"button_2":"","output_3":"","output_0":""}
    lock = threading.Lock()
    #schrijf hier eigen klas variabelen, zoals
    #Lists voor selects en radiogroups
    teller = 0

    @classmethod
    def start(cls):
        #schrijf hieronder je initialisatie code:
        #init gpio pinnen (RPi.GPIO), init sercomm ...
        print("start worker")

    @classmethod
    def stop(cls):
        #schrijf hieronder je opkuis code:
        #opkuisen gpio pinnen (RPi.GPIO), sluiten sercomm ...
        print("stop worker")

    @classmethod
    def do_btn_teller(cls):
        try:
            cls.lock.acquire()
            #schrijf hier code, bijv. getters en/of setters
            cls.teller += 1
            boodschap = "Er is reeds " + str(cls.teller) + " maal contact gemaakt."
            cls.set_txt_teller(boodschap)

        finally:
            cls.lock.release()

        return json.dumps(cls.data,ensure_ascii=False)

    @classmethod
    def get_btn_teller(cls):
        return cls.data["button_2"]

    @classmethod
    #maak dat een setter altijd tussen cls.lock.acquire()
    #en cls.lock.release() staat!!!!
    def set_btn_teller(cls,val):
        cls.data["button_2"]=val

    @classmethod
    def do_txt_teller(cls):
        try:
            cls.lock.acquire()
            #schrijf hier code, bijv. getters en/of setters
        finally:
            cls.lock.release()

        return json.dumps(cls.data,ensure_ascii=False)
```

```

@classmethod
def get_txt_teller(cls):
    return cls.data["output_3"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_txt_teller(cls,val):
    cls.data["output_3"]=val

@classmethod
def do_output_0(cls):
    try:
        cls.lock.acquire()
        #schrijf hier code, bijv. getters en/of setters
    finally:
        cls.lock.release()

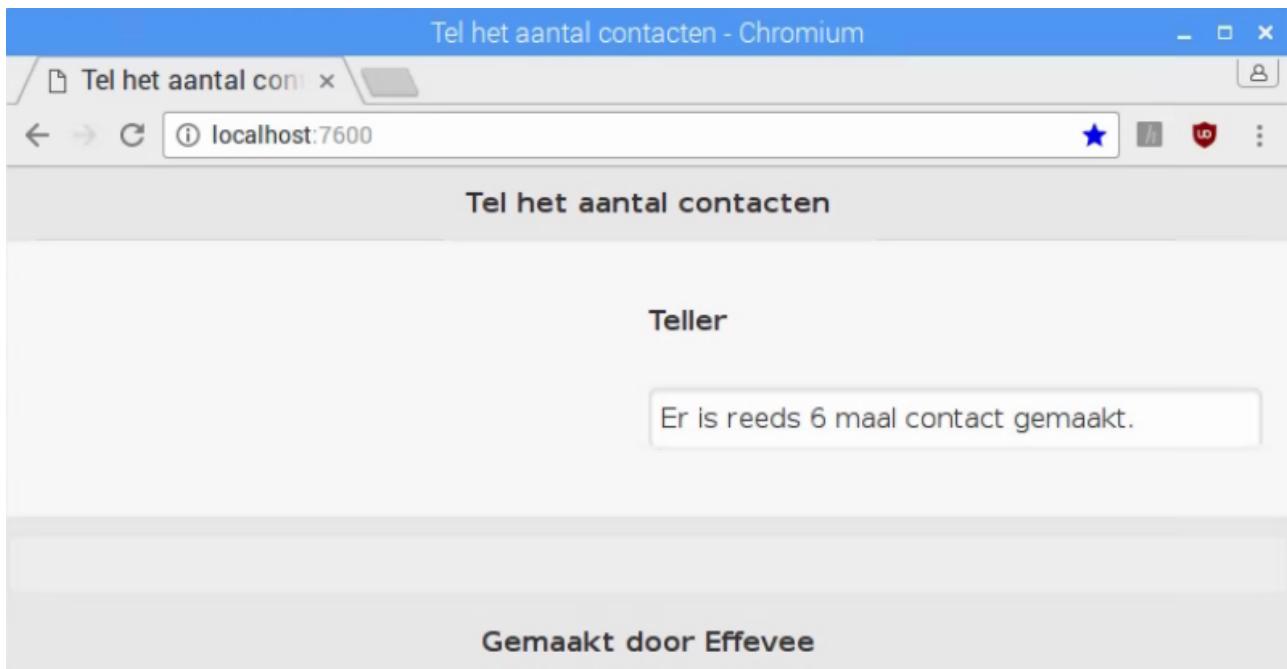
    return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_output_0(cls):
    return cls.data["output_0"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_output_0(cls,val):
    cls.data["output_0"]=val

```

## Webapplicatie GUI



# Project 21 : ESP8266 als remote mp3 speler

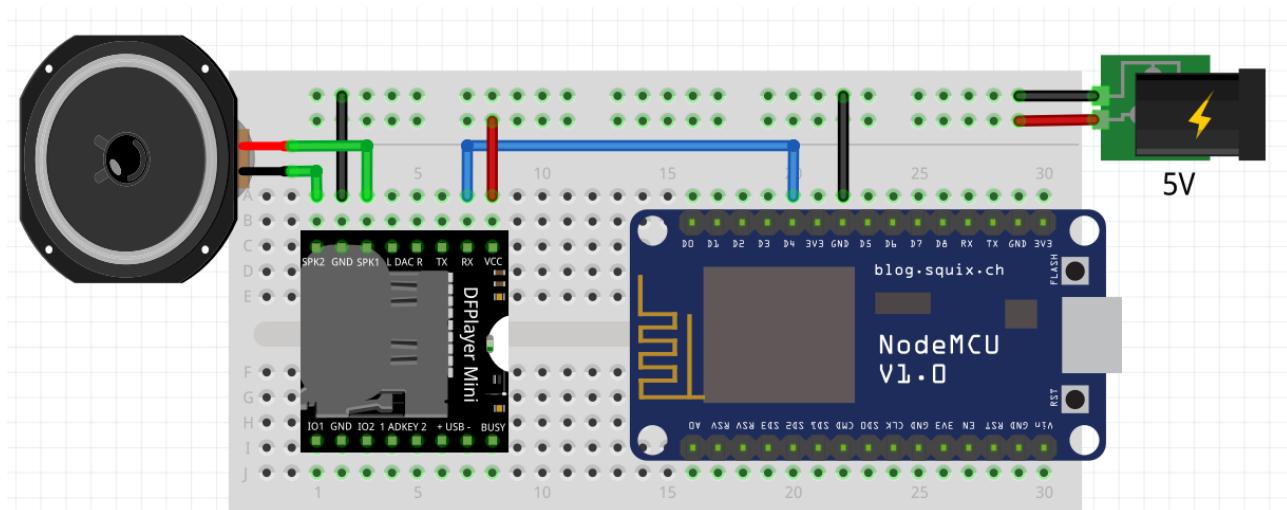
## Opgave

Maak van je ESP een mp3 speler die je van op afstand kan bedienen met een webinterface op je Raspberry Pi.

## Materiaal

- NodeMCU ESP8266 bordje
- DFR0299 mini mp3 player + micro SD kaart met mp3 files
- speaker
- externe voeding 5V

## Schakeling



DFR0299 Pin	Naar Pin	ESP8266 GPIO
VCC	Voeding +5V	-
RX	ESP8266 D4	TXD1
SPK1	Speaker	-
GND	Voeding GND ESP8266 GND	-
SPK2	Speaker	-

## DFR0299

De namen van de mp3 bestanden op de microSD kaart moeten allemaal beginnen met een nummer : vb. 0001liedje1.mp3, 0002liedje2.mp3, enz. Voor de seriële aansturing van de mp3player is slechts de TX poort van de ESP8266 nodig; de DFR0299 stuurt niets terug.

## Code

### *Mp3PlayerMod.py*

```
from machine import UART

class mp3Player:
    br = 9600
    TxPin = 1
    ser = None
    values = bytearray([0,0,0,0,0,0,0,0,0,0])

    @classmethod
    def start(cls):
        try:
            cls.ser = UART(cls.TxPin, cls.br)
            return True
        except Exception as e:
            print("problem connecting mp3player")
            print(e)
            return False

    @classmethod
    def command(cls):
        check = 0
        for j in range(2,8):
            check+=cls.values[j]
        check8 = check&0xFF
        check8Invert = 0xFF-check8
        cls.values[8] = check8Invert
        cls.ser.write(cls.values)

    @classmethod
    def play(cls):
        cls.values=bytearray([0X7E, 0xFF, 0x06, 0X0D, 00, 00, 00, 0xFE, 0xee, 0xEF])
        cls.command()

    @classmethod
    def playTrack(cls, track):
        try:
            cls.values = bytearray([0X7E, 0xFF, 0x06, 0X03, 00, 00, 00, 0xFE, 0xee, 0xEF])
            cls.values[5] = 0xFF&(track>>8)
            cls.values[6] = track&0xFF
            cls.command()
            return True
        except:
            return False

    @classmethod
    def pause(cls):
        cls.values = bytearray([0x7E, 0xFF, 0x06, 0x0E, 0x00, 0x00, 0x00, 0xFE, 0xED, 0xEF])
        cls.command()
```

```

@classmethod
def stop(cls):
    cls.values = bytearray([0x7E, 0xFF, 0x06, 0x16, 0x00, 0x00, 0x00,
0xFE, 0xED, 0xEF])
    cls.command()

@classmethod
def next(cls):
    cls.values = bytearray([0x7E, 0xFF, 0x06, 0x01, 0x00, 0x00, 0x00,
0xFE, 0xFA, 0xEF])
    cls.command()

@classmethod
def prev(cls):
    cls.values = bytearray([0x7E, 0xFF, 0x06, 0x02, 0x00, 0x00, 0x00,
0xFE, 0xF9, 0xEF])
    cls.command()

@classmethod
def volumeStepDown(cls):
    cls.values = bytearray([0x7E, 0xFF, 0x06, 0x05, 0x00, 0x00, 0x00,
0xFE, 0xED, 0xEF])
    cls.command()

@classmethod
def volumeStepUp(cls):
    cls.values = bytearray([0x7E, 0xFF, 0x06, 0x04, 0x00, 0x00, 0x00,
0xFE, 0xED, 0xEF])
    cls.command()

@classmethod
def setVolume(cls, volume):
    try:
        cls.values = bytearray([0x7E, 0xFF, 0x06, 0x06, 0x00, 0x00, 0x00,
0xFE, 0x00, 0xEF])
        cls.values[5] = 0xFF&(volume>>8)
        cls.values[6] = volume&0xFF
        cls.command()
        return True
    except:
        return False

```

### ***mp3PlayerESP.py***

```

from machine import Pin
from networking import wifi, socketServer
from mp3PlayerMod import mp3Player

def selectCommando(mp3, commando):
    print(commando)
    command = commando.split(":")
    if command[0] == 'play':
        if len(command) == 1:
            mp3.play()
        else:
            mp3.playTrack(int(command[1]))
    elif command[0] == 'pause':
        mp3.pause()

```

```

        elif command[0] == 'stop':
            mp3.stop()
        elif command[0] == 'next':
            mp3.next()
        elif command[0] == 'prev':
            mp3.prev()
        elif command[0] == 'vol+':
            mp3.volumeStepUp()
        elif command[0] == 'vol-':
            mp3.volumeStepDown()
        elif command[0] == 'vol':
            waarde = int(command[1])
            volume = int(waarde * 30.0 / 100.0)
            mp3.setVolume(volume)

def app():
    # connecteren met netwerk
    wifi.connectWIFI()
    if socketServer.start() == False:
        wifi.disconnectWIFI()
        return 0

    # initialisatie mp3 player
    mp3 = mp3Player()
    mp3.start()

    while True:
        status = socketServer.waitCn()
        if status < 0:
            print("Wrong command")
        elif status == 0:
            print("Halt ESP")
            break
        else:
            socketServer.sendData("OK")
            res = socketServer.readData()
            selectCommando(mp3, res)
            socketServer.sendData("OK")
            socketServer.stopClient()

    wifi.disconnectWIFI()

app()

```

### ***mp3 folder op microSD kaart***

```

frank@hirogen /media/frank/B6D1-AC52/mp3 $ ls
0001Coldplay - Fix You.mp3
0002U2 - Sunday Bloody Sunday.mp3
0003Of Monsters and Men - Little Talks.mp3
0004Birdy - Skinny Love.mp3
0005Simple Minds - Belfast Child.mp3
0006ABBA - The Way Old Friends Do.mp3

```

### **AppServers/MP3PlayerApp/webpy/playlist.txt**

(dubbel punt als scheiding tussen volgnummer en beschrijving mp3)

```
0001:Coldplay - Fix You
0002:U2 - Sunday Bloody Sunday
0003:Of Monsters and Men - Little Talks
0004:Birdy - Skinny Love
0005:Simple Minds - Belfast Child
0006:ABBA - The Way Old Friends Do
```

### **AppServers/MP3PlayerApp/webpy/worker.py**

```
import threading
import json
from networkingPI import PiSocketClient

class work:

    data={"button_9":"","button_8":"","slider_11":"0","button_3":"","button_2":"",
    "button_4":"","button_7":"","button_6":"","select_5":"","button_10":"","button_
    _12":"","output_0":""}
    lock = threading.Lock()
    #schrijf hier eigen klas variabelen, zoals
    #Lists voor selects en radiogroups
    SERVER = "192.168.1.30"
    PORT = 1022
    PLAYLIST = []
    TRACK = 1

    @classmethod
    def start(cls):
        #schrijf hieronder je initialisatie code:
        #init gpio pinnen (RPi.GPIO), init sercomm ...
        print("start worker")
        # opvullen playlist ahv tekstfile
        with open("playlist.txt") as file:
            for line in file:
                line = line.strip()
                cls.PLAYLIST.append(line)
        # opvullen select met songs van PLAYLIST
        cls.data["select_5"] =
    {"fill@select":cls.PLAYLIST,"value":cls.PLAYLIST[0]}
        # volume op helft
        cls.do_sld_volume("50")

    @classmethod
    def stop(cls):
        #schrijf hieronder je opkuis code:
        #opkuisen gpio pinnen (RPi.GPIO), sluiten sercomm ...
        print("stop worker")
```

```

@classmethod
def do_btn_voldown(cls):
    # volume verlagen
    volume = int(cls.get_sld_volume())
    volume -= 1
    if volume < 0:
        volume = 0
    cls.do_sld_volume(str(volume))

    return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_btn_voldown(cls):
    return cls.data["button_10"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_btn_voldown(cls,val):
    cls.data["button_10"]=val

@classmethod
def do_btn_volup(cls):
    # volume verhogen
    volume = int(cls.get_sld_volume())
    volume += 1
    if volume > 100:
        volume = 100
    cls.do_sld_volume(str(volume))

    return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_btn_volup(cls):
    return cls.data["button_12"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_btn_volup(cls,val):
    cls.data["button_12"]=val

@classmethod
def do_output_0(cls):
    try:
        cls.lock.acquire()
        #schrijf hier code, bijv. getters en/of setters
    finally:
        cls.lock.release()

    return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_output_0(cls):
    return cls.data["output_0"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_output_0(cls,val):
    cls.data["output_0"]=val

```

```

@classmethod
def do_btn_pauze(cls):
    try:
        cls.lock.acquire()
        #schrijf hier code, bijv. getters en/of setters
        if PiSocketClient.start(cls.SERVER, cls.PORT):
            PiSocketClient.sendData("go")
            res = PiSocketClient.readData()
            PiSocketClient.sendData("pause")
            res = PiSocketClient.readData()
            PiSocketClient.stop()
    finally:
        cls.lock.release()

    return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_btn_pauze(cls):
    return cls.data["button_3"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_btn_pauze(cls,val):
    cls.data["button_3"]=val

@classmethod
def do_btn_play(cls):
    try:
        cls.lock.acquire()
        #schrijf hier code, bijv. getters en/of setters
        if PiSocketClient.start(cls.SERVER, cls.PORT):
            PiSocketClient.sendData("go")
            res = PiSocketClient.readData()
            PiSocketClient.sendData("play")
            res = PiSocketClient.readData()
            PiSocketClient.stop()
    finally:
        cls.lock.release()

    return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_btn_play(cls):
    return cls.data["button_2"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_btn_play(cls,val):
    cls.data["button_2"]=val

@classmethod
def do_btn_stop(cls):
    try:
        cls.lock.acquire()
        #schrijf hier code, bijv. getters en/of setters
        if PiSocketClient.start(cls.SERVER, cls.PORT):
            PiSocketClient.sendData("go")
            res = PiSocketClient.readData()
            PiSocketClient.sendData("stop")
            res = PiSocketClient.readData()
            PiSocketClient.stop()

```

```

        finally:
            cls.lock.release()

        return json.dumps(cls.data,ensure_ascii=False)

    @classmethod
    def get_btn_stop(cls):
        return cls.data["button_4"]

    @classmethod
    #maak dat een setter altijd tussen cls.lock.acquire()
    #en cls.lock.release() staat!!!!
    def set_btn_stop(cls,val):
        cls.data["button_4"]=val

    @classmethod
    def do_btn_next(cls):
        try:
            cls.lock.acquire()
            #schrijf hier code, bijv. getters en/of setters
            if PiSocketClient.start(cls.SERVER, cls.PORT):
                PiSocketClient.sendData("go")
                res = PiSocketClient.readData()
                # volgende track
                cls.TRACK += 1
                if cls.TRACK > len(cls.PLAYLIST):
                    cls.TRACK = 1
                cls.set_sel_track(cls.PLAYLIST[cls.TRACK-1])
                PiSocketClient.sendData("play:"+str(cls.TRACK))
                # PiSocketClient.sendData("next")
                res = PiSocketClient.readData()
                PiSocketClient.stop()
        finally:
            cls.lock.release()

        return json.dumps(cls.data,ensure_ascii=False)

    @classmethod
    def get_btn_next(cls):
        return cls.data["button_8"]

    @classmethod
    #maak dat een setter altijd tussen cls.lock.acquire()
    #en cls.lock.release() staat!!!!
    def set_btn_next(cls,val):
        cls.data["button_8"]=val

    @classmethod
    def do_btn_prev(cls):
        try:
            cls.lock.acquire()
            #schrijf hier code, bijv. getters en/of setters
            if PiSocketClient.start(cls.SERVER, cls.PORT):
                PiSocketClient.sendData("go")
                res = PiSocketClient.readData()
                # vorige track
                cls.TRACK -= 1
                if cls.TRACK <= 0:
                    cls.TRACK = len(cls.PLAYLIST)
                cls.set_sel_track(cls.PLAYLIST[cls.TRACK-1])
                PiSocketClient.sendData("play:"+str(cls.TRACK))

```

```

        # PiSocketClient.sendData("prev")
        res = PiSocketClient.readData()
        PiSocketClient.stop()
    finally:
        cls.lock.release()

    return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_btn_prev(cls):
    return cls.data["button_7"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_btn_prev(cls,val):
    cls.data["button_7"]=val

@classmethod
def do_btn_first(cls):
    try:
        cls.lock.acquire()
        #schrijf hier code, bijv. getters en/of setters
        if PiSocketClient.start(cls.SERVER, cls.PORT):
            PiSocketClient.sendData("go")
            res = PiSocketClient.readData()
            # eerste track
            cls.TRACK = 1
            cls.set_sel_track(cls.PLAYLIST[cls.TRACK-1])
            PiSocketClient.sendData("play:"+str(cls.TRACK))
            # PiSocketClient.sendData("play:1")
            res = PiSocketClient.readData()
            PiSocketClient.stop()
    finally:
        cls.lock.release()

    return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_btn_first(cls):
    return cls.data["button_6"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_btn_first(cls,val):
    cls.data["button_6"]=val

@classmethod
def do_btn_last(cls):
    try:
        cls.lock.acquire()
        #schrijf hier code, bijv. getters en/of setters
        if PiSocketClient.start(cls.SERVER, cls.PORT):
            PiSocketClient.sendData("go")
            res = PiSocketClient.readData()
            # laatste track
            cls.TRACK = len(cls.PLAYLIST)
            cls.set_sel_track(cls.PLAYLIST[cls.TRACK-1])
            PiSocketClient.sendData("play:"+str(cls.TRACK))
            res = PiSocketClient.readData()
            PiSocketClient.stop()

```

```

        finally:
            cls.lock.release()

        return json.dumps(cls.data,ensure_ascii=False)

    @classmethod
    def get_btn_last(cls):
        return cls.data["button_9"]

    @classmethod
    #maak dat een setter altijd tussen cls.lock.acquire()
    #en cls.lock.release() staat!!!!
    def set_btn_last(cls,val):
        cls.data["button_9"]=val

    @classmethod
    def do_sld_volume(cls,val):
        try:
            cls.lock.acquire()
            cls.set_sld_volume(val)
            #schrijf hier code, bijv. aan de hand van val een LED doen
branden
            if PiSocketClient.start(cls.SERVER, cls.PORT):
                PiSocketClient.sendData("go")
                res = PiSocketClient.readData()
                PiSocketClient.sendData("vol:"+val)
                res = PiSocketClient.readData()
                PiSocketClient.stop()
        finally:
            cls.lock.release()

        return json.dumps(cls.data,ensure_ascii=False)

    @classmethod
    def get_sld_volume(cls):
        return cls.data["slider_11"]

    @classmethod
    #maak dat een setter altijd tussen cls.lock.acquire()
    #en cls.lock.release() staat!!!!
    def set_sld_volume(cls,val):
        cls.data["slider_11"]=val

    @classmethod
    def do_sel_track(cls,val):
        try:
            cls.lock.acquire()
            cls.set_sel_track(val)
            #schrijf hier code, bijv. aan de hand van val een LED doen
branden
            if PiSocketClient.start(cls.SERVER, cls.PORT):
                PiSocketClient.sendData("go")
                res = PiSocketClient.readData()
                print(val)
                track = val.split(":")
                cls.TRACK = int(track[0])
                PiSocketClient.sendData("play:"+str(cls.TRACK))
                res = PiSocketClient.readData()
                PiSocketClient.stop()
        finally:
            cls.lock.release()

```

```

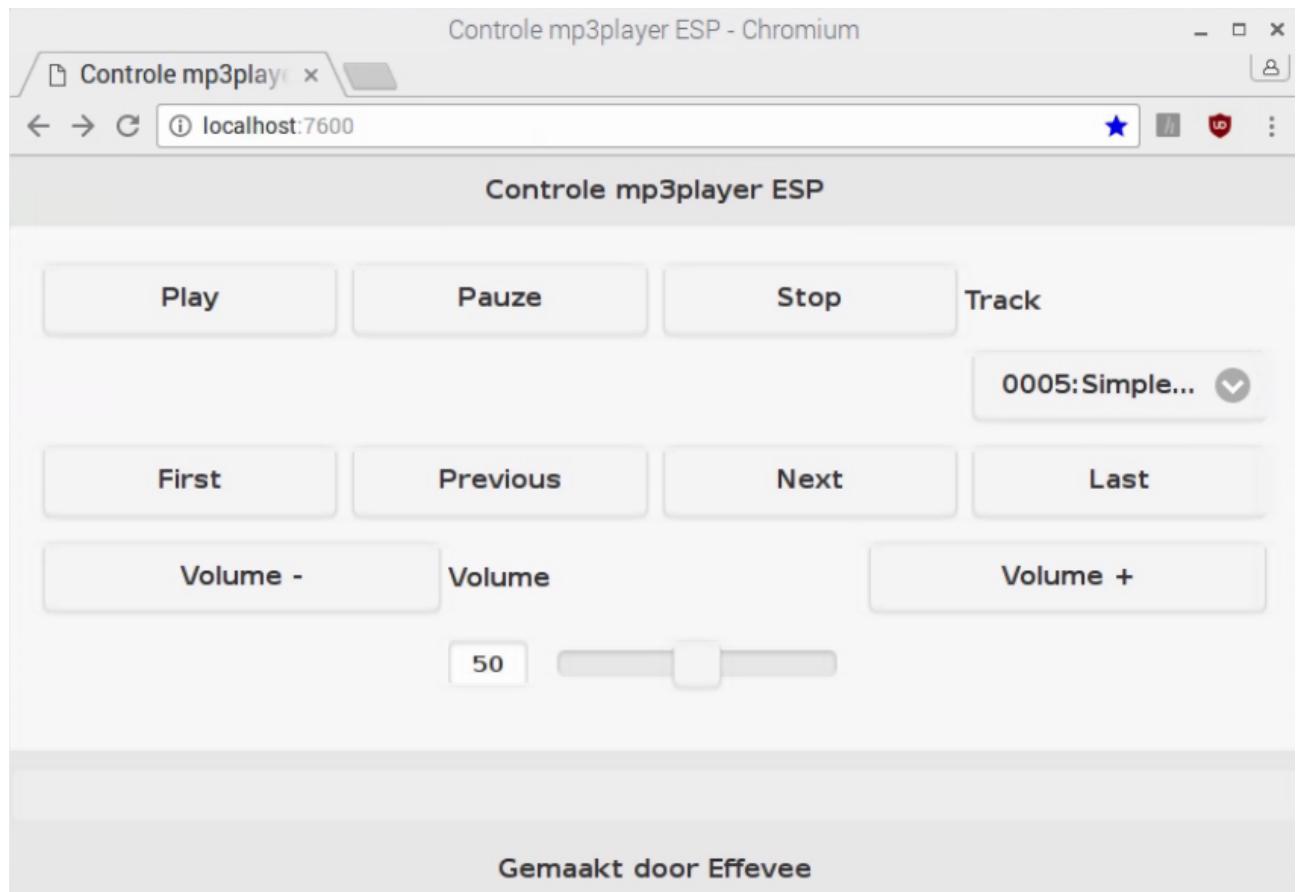
        return json.dumps(cls.data,ensure_ascii=False)

@classmethod
def get_sel_track(cls):
    return cls.data["select_5"]

@classmethod
#maak dat een setter altijd tussen cls.lock.acquire()
#en cls.lock.release() staat!!!!
def set_sel_track(cls,val):
    cls.data["select_5"] = {"fill@select":cls.PLAYLIST,"value":val}

```

## Webapplicatie GUI



# Datasheets

## ESP8266

De datasheet van de ESP8266 WiFi microcontroller van Espressif Systems vind je [hier](#).

## Tower Pro MG90S

De datasheet voor de micro servo Tower Pro MG90S kan je [hier](#) vinden.

## 28BYJ-48

De datasheet van de 28BYJ-48 stappenmotor van Kiatronics vind je [hier](#).

## ULN200xA

De datasheet van de ULN200xA Darlington array IC van Diodes Incorporated kan je [hier](#) vinden.

## L298

De datasheet van de dual H bridge L298 van ST Microelectronics vind je [hier](#).

## DHT11 / DHT22

De datasheet van de DHT11 temperatuur- en vochtigheidsmeter vind je [hier](#). Deze van de meer nauwkeurige DHT22 is [hier](#) te vinden.

## LDR

De datasheet van de Light Dependent Resistor (LDR) van Sunrom Technologies vind je [hier](#).

## MCP3004/3008

De datasheet van de 10bit analoge naar digitale converter (ADC) van Microchip vind je [hier](#).

## MAX7219

De datasheet van de MAX7219 seriële 8 digit LED display driver van Maxim kan je [hier](#) vinden.

## SSD1306

De datasheet van de ssd1306 OLED/PLED controller van Solomon Systech kan je [hier](#) vinden.

## HD44780U

De datasheet van de HD44780U dot matrix LCD display controller van Hitachi vind je [hier](#).

## PCF8574

De datasheet van de remote 8-bit I/O expander voor de I<sup>2</sup>C bus van Philips vind je [hier](#).

## **WS2812B**

De datasheet van de WS2812B adresseerbare RGB leds (NeoPixels) van Worldsemi vind je [hier](#).

## **DFR0299**

De datasheet van de mini mp3 player DFR0299 kan je [hier](#) vinden.

# Interessante websites

- [MicroPython homepage](#)
- [MicroPython firmware voor de ESP8266](#)
- [MicroPython op de ESP8266 documentatie](#)
- [MicroPython op de ESP8266 workshop](#)
- [MicroPython op de Wemos D1 Mini workshop](#)
- [Adafruit Learning System MicroPython guides](#)
- [ESP8266 stepper motor control with ULN2003 driver module](#)
- [DFPlayer Mini wiki](#)