Teknologisæt til Sæsonen 2021-2022
(Alle hold fra 10 år og op)

Igen i år kan vi give alle deltagere på alle vores hold i alle afdelinger et teknologisæt med hjem via vores projekt “Udvidelse af Teknologiskolen” under Science i fritiden hos Villumfonden.

Årets Teknologisæt er lavet med microcontrollerplatformen, Raspberry Pi Pico.

Pico Pi kan programmeres på flere forskellige måder. Den kommer senere til at kunne bruges sammen med Microsoft Makecode, som er det visuelle programmeringsværktøj vi også bruger til micro:bit. Det virker bare ikke ret godt endnu, og derfor vil alt materiale vi laver til denne sæsons sæt tage udgangspunkt i programmeringssproget, Python, der er tekstbaseret.

Installation

For at kunne arbejde med Python og Pico Pi skal man have installeret et programmeringsmiljø (IDE) - en app på din computer.

Vi bruger programmet, **Thonny**, der kan fungere både til Mac, Windows, Linux og Chromebooks

Thonny kan hentes til din computer her:
https://thonny.org

Der er instruktioner og downloads på siden til at installere det til Mac, Windows og Linux.

OBS Nedenstående instruktioner til Chromebook virker ikke med Chrome OS 100 og derover lige nu…

Hvis du skal have det til at virke på en Chromebook, skal du først have sat Chrome OS til at understøtte Linux, som beskrevet her:

1. I indstillinger er der noget der hedder "Linux (Beta)". Tryk på "Turn On" og installer det derefter. Det kan tage et stykke tid, så vær tålmodig. Når det er færdigt, åbner Linux-terminalen automatisk.
2. I terminalen skriver du "sudo apt-get update" og trykker enter for at få den seneste opdatering.
3. Når opdateringen er fuldført, kan du installere den nyeste version af Thonny med “bash <(wget -O - https://thonny.org/installer-for-linux)” og trykke enter.
4. Når overførslen er færdig, kan du lukke terminalen.

Du vil nu kunne finde Thonny på din startskærm.

Youtube og Pi Pico

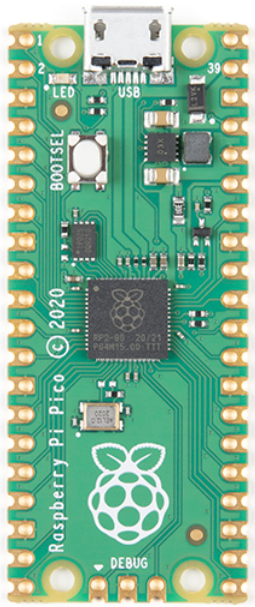
En god youtuber til Pi Pico og micropython: Kevin McAleer:
https://www.youtube.com/channel/UCuoS-cgppnO46VCcQi81jvQ

MicroPython til Raspberry Pi Pico

Oversat fra denne: Getting started with Raspberry Pi Pico - Introduction

1. Før du kan gå igang med at bruge Thonny sammen med din Pi Pico, skal der et specielt styre-program over på picoen: Brug linket for at hente det specielle program og gem det på din computer:
   https://micropython.org/resources/firmware/rp2-pico-20220117-v1.18.uf2

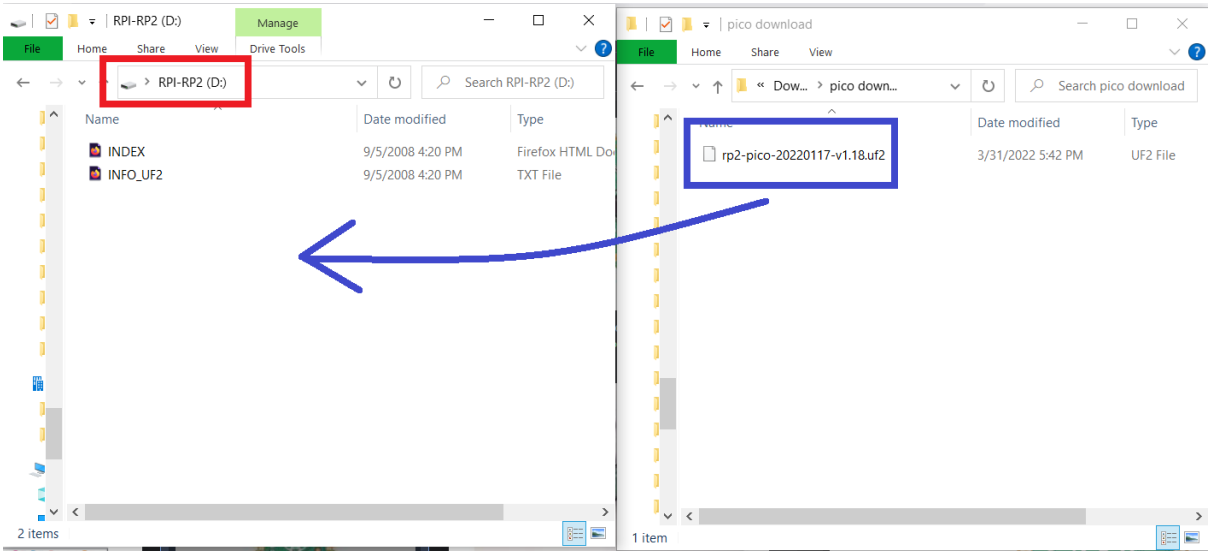
2. På Pico tryk på BOOTSEL (den hvide knap) og hold den inde, mens du sætter usb-kablet ind i picoen og ind i din computer. Det er vigtigt, at du først giver slip på den hvide knap, når pico'en er tilsluttet din computer.



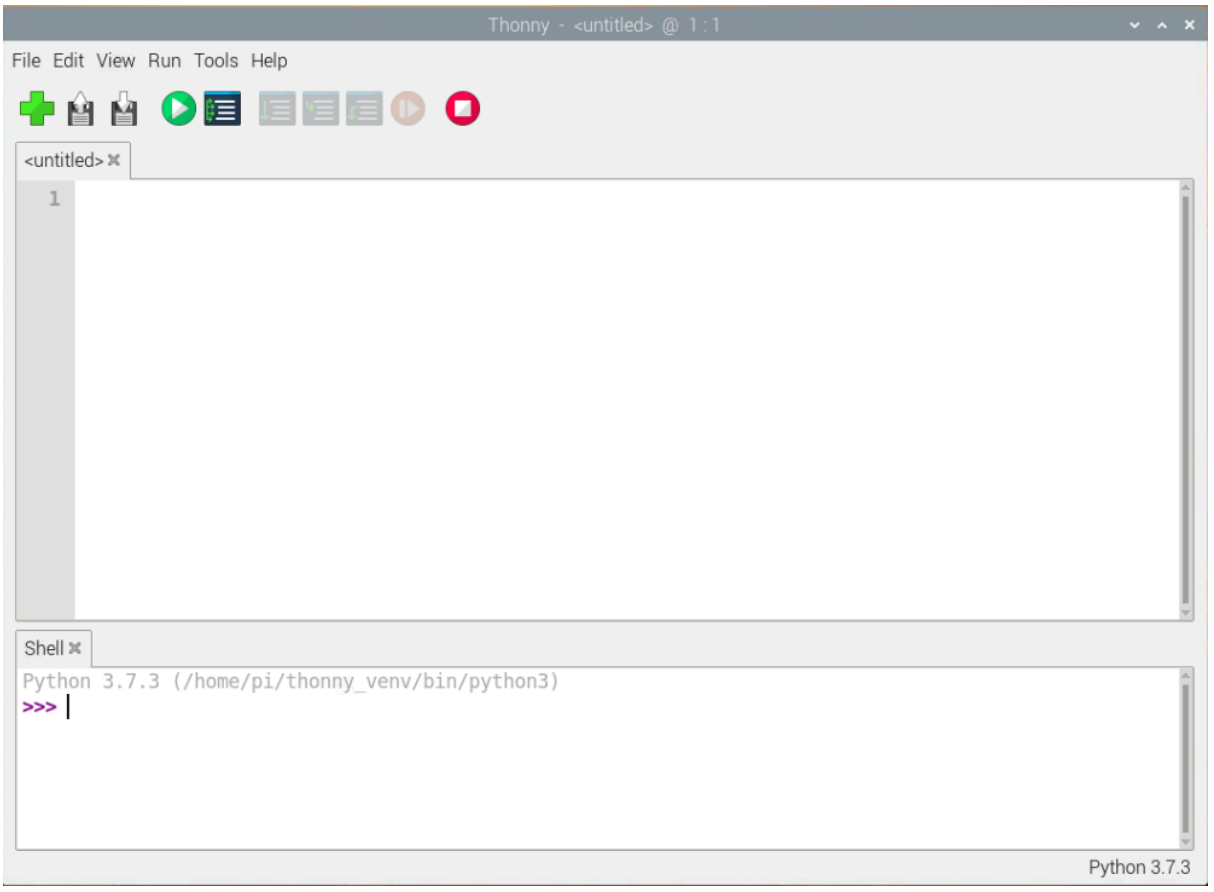
g

3. Picoen åbner et drev på din computer ligesom et usb-stik (rød) og du kan åbne den som en mappe på din computer. Nu skal du flytte det program, som du downloadede

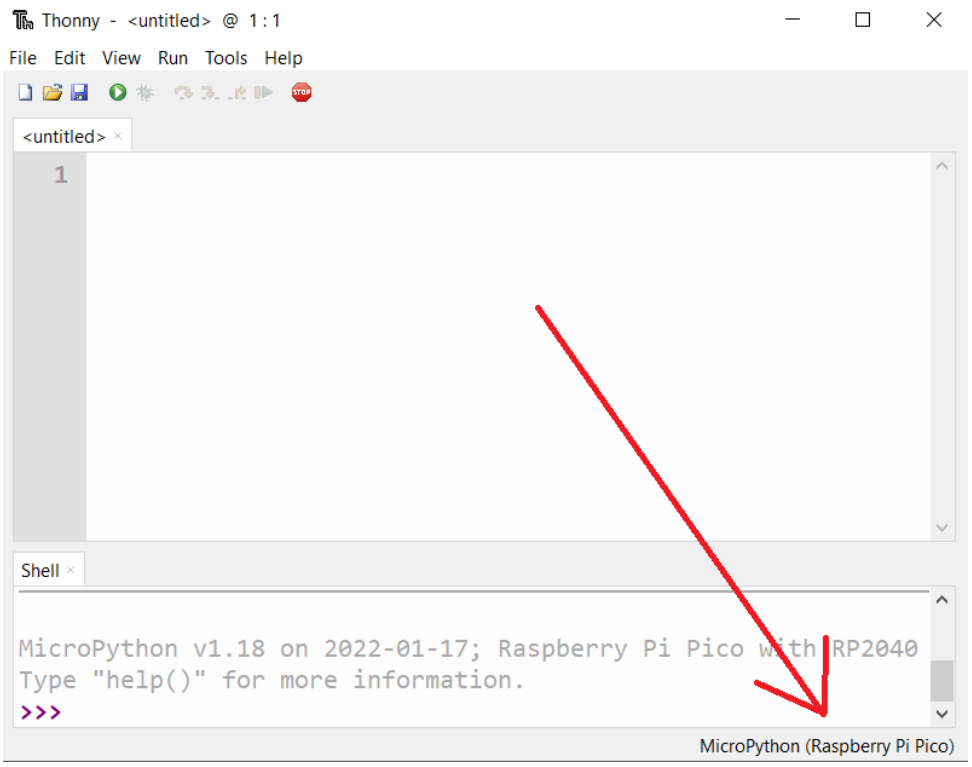
fra din computer over på picoen igesom på billedet (blå).k



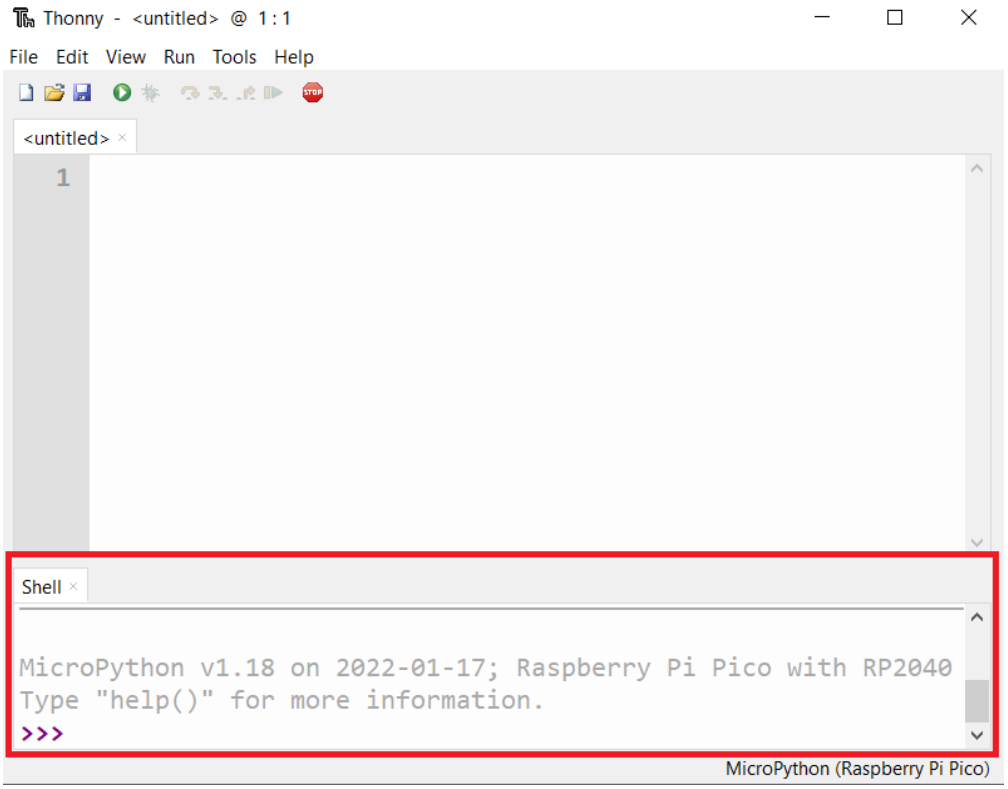
4. Download og installer Thonny fra linket ovenstående. Åbn det på din computer, det ser ud som på billedet:



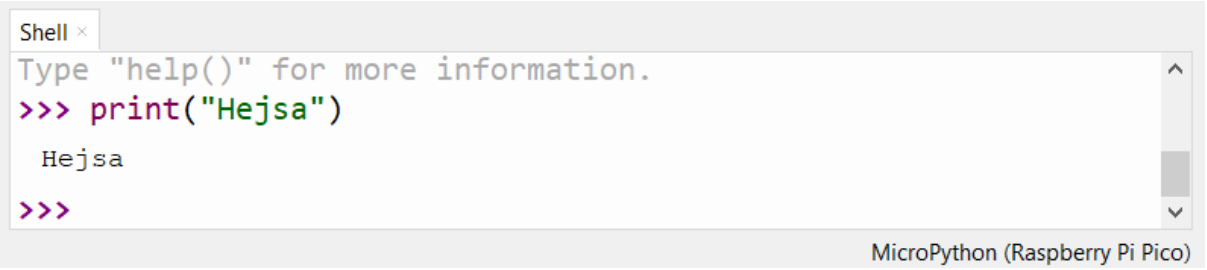
5. Klik på teksten i nederste højre hjørne og vælg “MicroPython (Raspberry Pi Pico)”



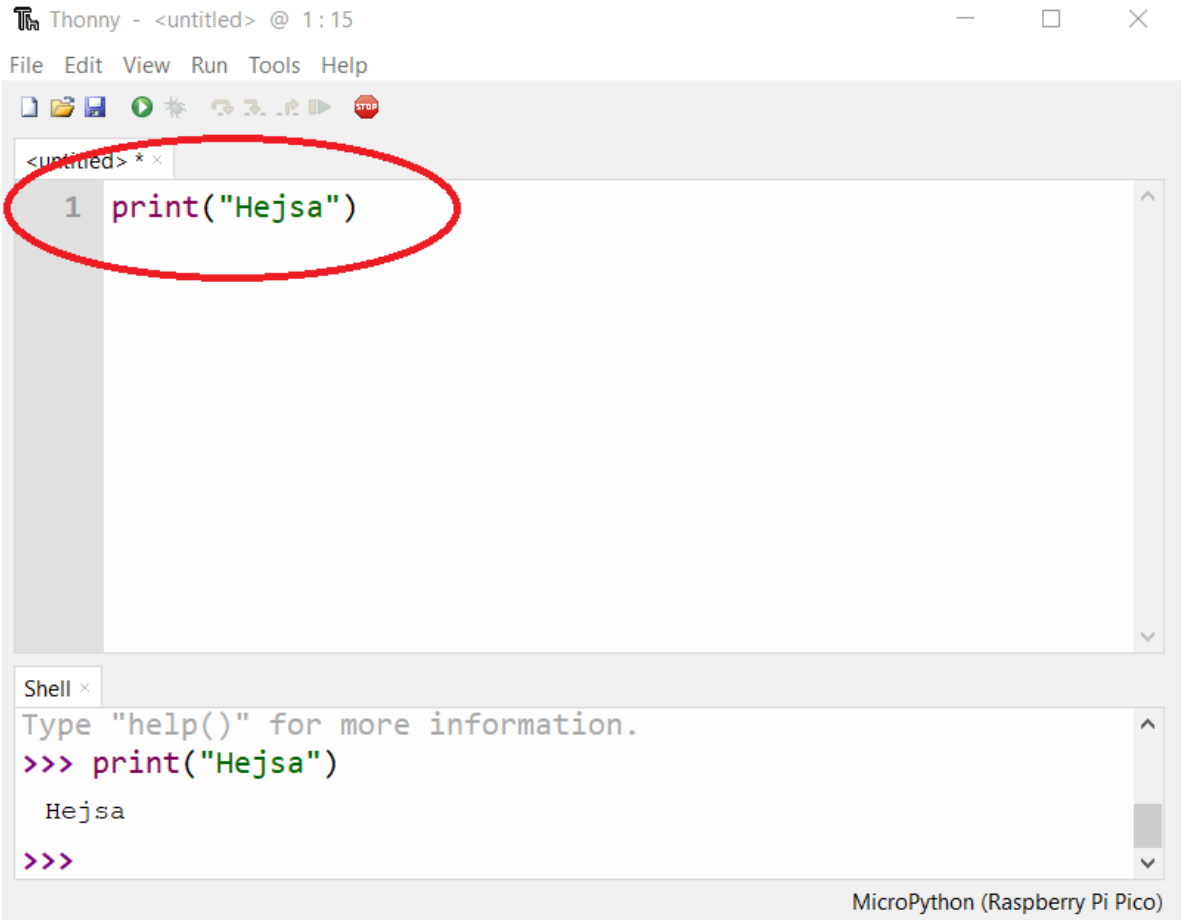
6. Nu burde din “Shell” se sådan her ud som markeret på det næste billede:



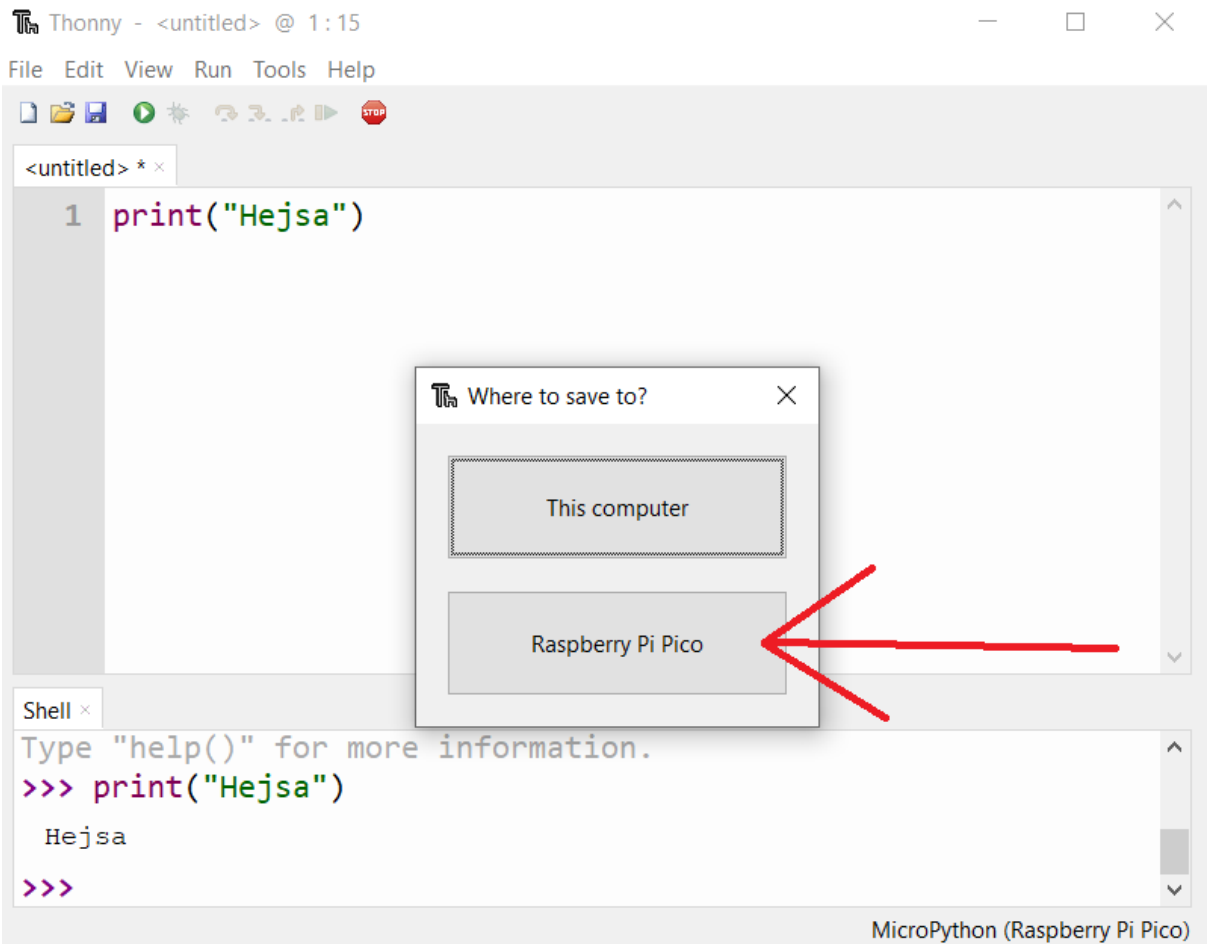
7. “Shell” er et sted, hvor man kan skrive meget små programmer og bede picoen om at lave nogle helt simple ting. Lad os prøve det af ved at skrive følgende ud fra de tre pile: **print(“Hejsa”)** og tryk så på enter-knappen. Resultatet ser sådan her ud:



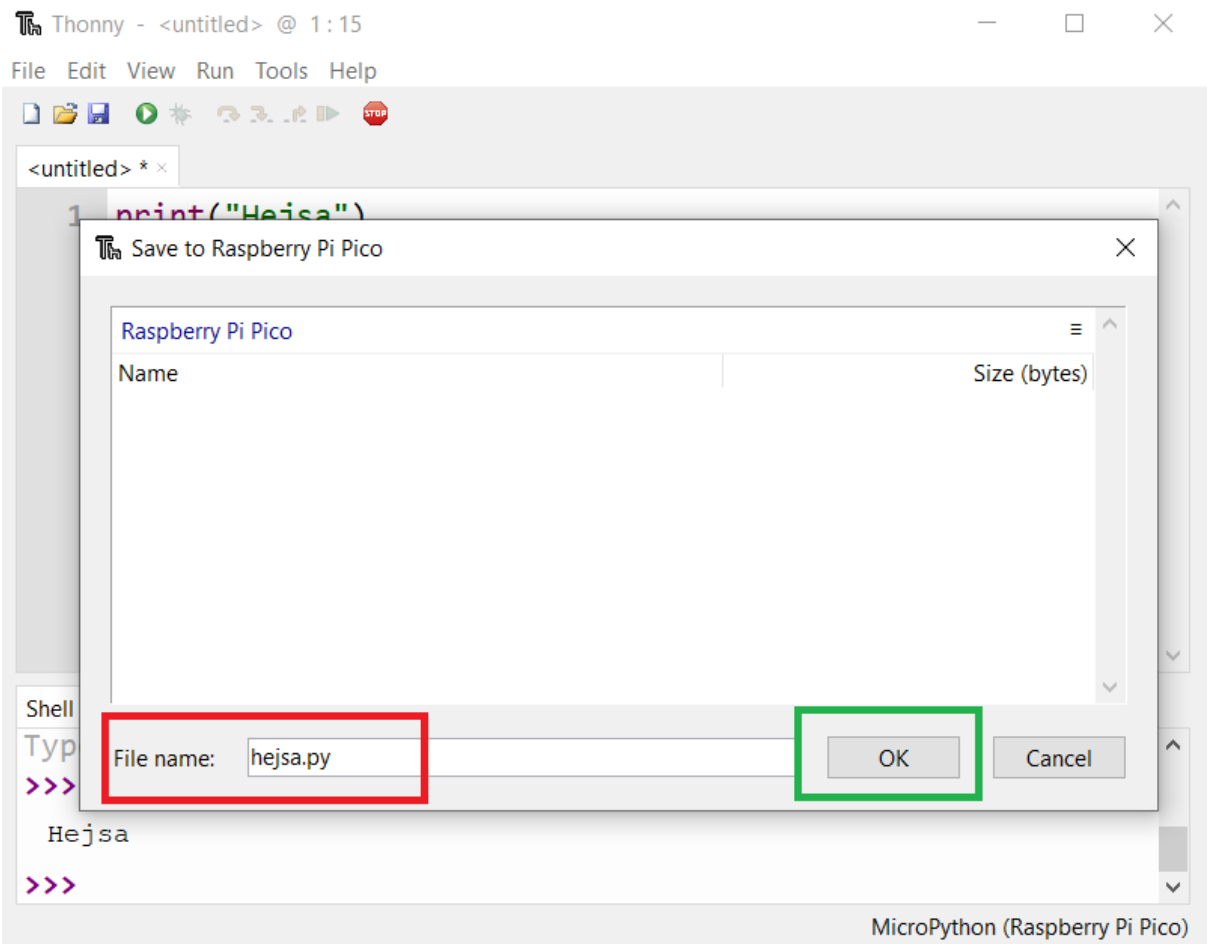
8. Tillykke, nu har du fået picoen til at sige hej til dig.
   Når vi vil lave mere avancerede programmer skal vi gemme alt vores kode i en fil.
   Prøv at skrive **print("Hejsa")** der, hvor der er et 1-tal, så har vi faktisk skrevet den allerførste linje i vores fil. Det er filen, som bliver til vores program.



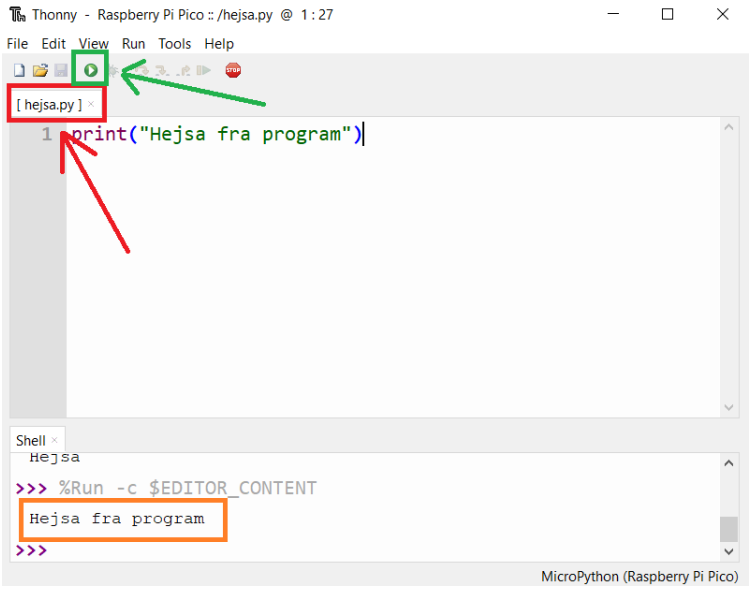
9. Tryk på tasterne **Ctrl** og **s** samtidig på din computer for at gemme filen. Vælg at gemme den på Raspberry Pi Pico ved at trykke på den nederste af de to muligheder ligesom på billedet:



10. Giv dit program en navn fx “hejsa.py” (rød boks) - Du kan kalde det hvad du har lyst, men du kan ikke bruge mellemrum. Du skal også altid huske, at filen skal slutte med “.py”. Tryk på ok (grøn boks), når har fundet et godt navn. Programmet bliver så gemt på Picoen:



11. Nu er programmet gemt på din pico og har fået et navn (rød boks). Prøv at ændre i teksten inde i **print()** og tryk derefter på play-knappen (grøn boks), så starter programmet og den nye tekst bliver vist nede i "Shell" (orange):



Lidt mere Python programmering

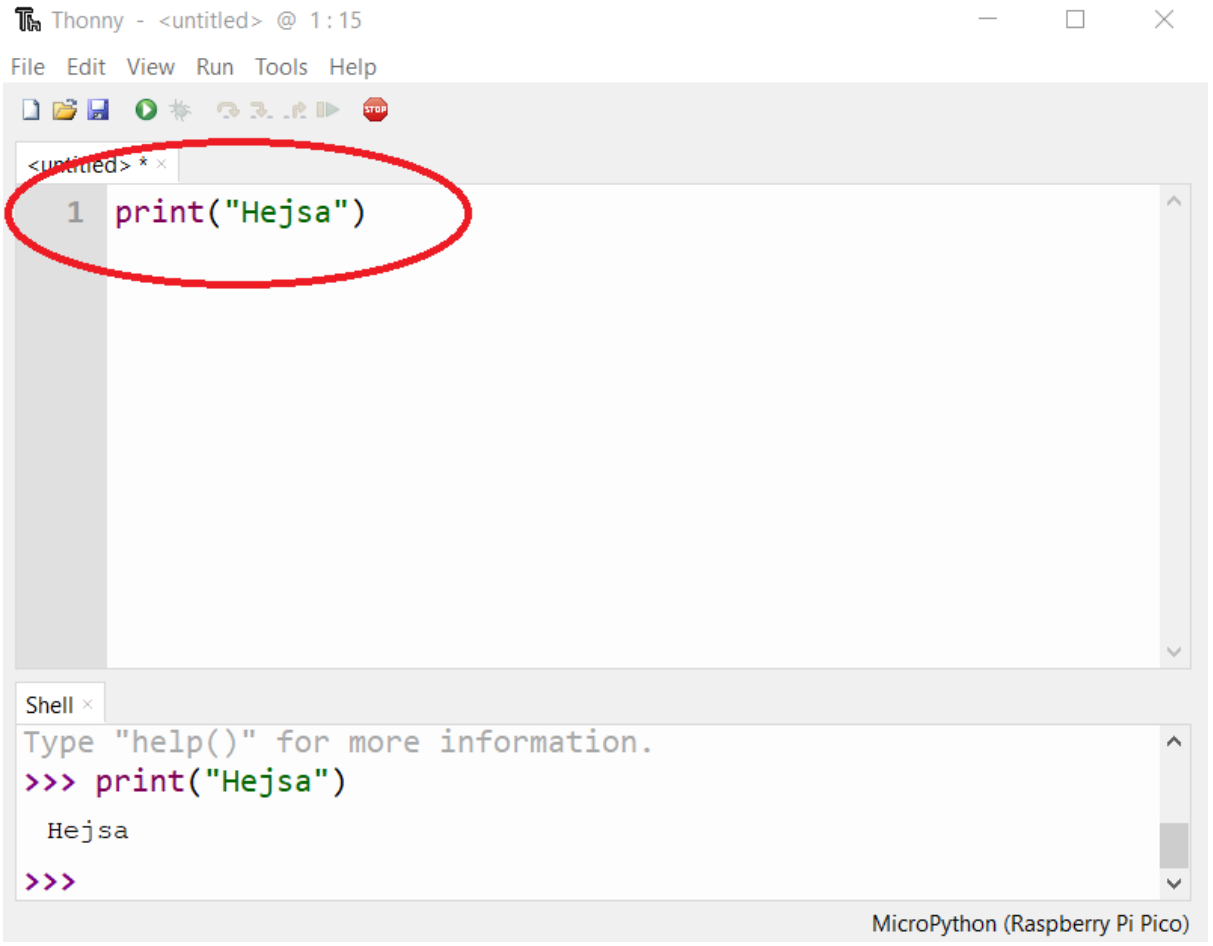
Der findes en masse bøger på engelsk, der kan lære dig om Python programmeringssproget. Der er bøger både for begyndere og hvis man kender til programmering i forvejen. Og der er bøger for børn og voksne.

På dansk findes der denne net-bog, som giver en god introduktion til Python programmering. Den handler generelt om Python sproget og altså ikke særligt om Python programmering og picoen:

https://pyprog.dk

Til at starte med er det vigtigt at du lærer sproget lidt at kende, og så viser vi bagefter, hvordan du kan begynde at læse inputs og skrive til outputs på picoen, så vi kan begynde at bygge små værktøjer og robotter.

Først fortsætter vi lige lidt med `print("Hejsa")` programmet fra før.



Vi bruger den indbyggede funktion, **print()** til at udskrive den tekststreng vi taster ind som **parameter** til funktionen. Alle funktioner har et sæt parenteser, **()** efter funktionsnavnet. Man skriver parametrene til funktionerne inde i parenteserne.

En parameter kan f.eks. være en længde. Lad os sige vi har en funktion, der beregner arealet af et kvadrat, ud fra kvadratets sidelængde. Så kunne funktionen se sådan ud:

```
kvadrat_areal(sidelængde)
```

Nogle funktioner har brug for mere end en parameter, og så adskilles parametrene med et komma. F.eks. **funktion(parameter_1, parameter_2).**

Parametrene kan være forskellige typer. Print skal f.eks. have en tekststreng som sin første parameter. Tekststrenge kan man kende ved at de er "pakket ind i" et sæt gåseøjne, **"tekststreng"**.

Hvis man vil vide mere om hvilke parametre en bestemt funktion bruger kan man slå det op i Python dokumentationen. Se f.eks. alle Pythons indbyggede funktioner her:

https://www.w3schools.com/python/python_ref_functions.asp

Kan du finde **print()** - funktionen og se, hvad man skal skrive hvis man gerne vil have den til at udskrive to tekststrenge med en **print()** - funktion?

Loops og delays

Lad os starte med et **for**-loop:

```
print("Start loopet!")
for i in range(10):
  print("Loop nr ", i)
print("Loop slut!")
```

Prøv at taste programmet ind i Thonny og kør det på picoen.

Læg mærke til at Python tæller fra 0. Derfor når den kun op til 9, når man skriver at den skal loope 10 gange.

I Python er det vigtigt at man rykker de linjer ind, der hører til et loop eller en if-sætning. ligesom `print("Loop nr ", i)` linjen i ovenstående program.

"i" er en variabel, der automatisk oprettes når man skriver `for i in range(10):` Variablen tælles een op hver gang loopet kører en runde, og man kan kalde "i"-variablen inde i loopet, som vi gør, når vi skriver den ud med `print("Loop nr ", i)`

Læg mærke til at **print()** - funktionen her bruger 2 parametre, og at den selv kan finde ud af at oversætte tal-variablen "i" til tekst, så det kan skrives ud sammen med teksten "Loop nr ".

Vi kan også lave nogle pauser i vores program, så det ikke går så hurtigt.

For at gøre det skal man importere python biblioteket **utime**:

```
import utime

print("Start loopet!")
for i in range(10):
  print("Loop nr", i)
  utime.sleep(1)
print("Loop slut!")
```

Man skal importere de biblioteker man skal bruge i toppen af sit program. Det kan være biblioteker,som **utime**, der har forskellige funktioner til at styre tid, eller f.eks. biblioteker til at styre noget bestemt hardware, som Neopixels eller motorer.

Man kan også bruge `while True:` til at loope. Det fungerer på samme måde som **for altid** loopet i Makecode.

```
import utime

print("Start loopet!")
while True:
  print("Loopet kører!")
  utime.sleep(1)
print("Loop slut!")
```

Programmet stopper aldrig - tryk på Stop i Thonny, for at afbryde programmet.

Variabler

En variabel er en slags beholder til at gemme noget i. I programmering bruges variabler til at gemme noget information som man skal bruge senere i programmet. Det kan f.eks. være et tidspunkt man skal huske eller en farve eller en masse andet. Variabler kan både indeholde tekst og tal.

Man kan tænke på dem som en kasse man gemmer noget i - uden på kassen skriver man noget, der fortæller hvad det er man gemmer. Tænk f.eks. på en flyttekasse. Her skriver man f.eks. “køkken” uden på den kasse der indeholder ting fra køkkenet, så man lettere kan finde den igen, når man skal til at sætte ting på plads.
De fleste variabler indeholder dog kun en ting - enten et tal eller noget tekst. Tekst kalder man også for en streng i programmering.

Hvis vi f.eks. vil gemme et tal i en variabel kan vi skrive følgende:

```
tal = 7
```

Hvis vi vil gemme tekst kan vi i stedet skrive:

```
tekst = "Super Duper"
```

Hvis vi så senere i programmet gerne vil bruge vores variabler igen kan vi f.eks. skrive

```
print(tal)
print(tekst)
```

Og så bliver følgende skrevet ud

```
7
Super Duper
```

Man kan sagtens ændre indholdet af en variabel igen senere i programmet, f.eks. kan vi skrive `tal = 10` og så vil vi få 10 skrevet ud, hvis vi skriver `print(tal)`

Variabelnavne skrives næsten altid med tekst, og må ikke starte med tal. Man kan altså godt skrive `var40 = 13`, men ikke `40var = 13`. Variabelnavne må heller ikke indeholde mellemrum, men mange bruge underscore i stedet for: `mit_tal = 20`.

Det kan være en rigtig god ide at give så beskrivende variabelnavne som muligt i éns programmer. F.eks.

```
bil_hastighed = 20 #eller
bil_status = "i stykker"
```

Betingelser og Løkker (loops)

Betingelser er en måde i programmering, hvor man kan få programmet til at gøre noget bestemt, hvis en betingelse er opfyldt - altså f.eks. hvis lyset er slukket, så tænd det. Vi kender det fra andre programmeringssprog også - som if / else og hvis / ellers og det er også de kommandoer vi bruger i python:

Prøv f.eks. følgende i Thonny:

```
navn = input ("Hvad er dit navn? ")
if navn == "Batman":
```

```
    print("Du er Batman!")
else:
    print("Du er ikke Batman!")
```

Det man giver som input til en if-sætning, skal altså være noget der enten giver sandt eller falsk - altså f.eks. `if alder >= 25:` eller `if farve == "gul":`

Læg mærke til at man sammenligner lighed med 2 lighedstegn efter hinanden "==" i python. Følgende operatorer kan bruges til sammenligninger:

- == Lig med
- != Forskellig fra
- > Større end
- >= Større end eller lig med
- < Mindre end
- <= Mindre end eller lig med

Husk at slutte en if-linje med et kolon (:), og i python skal man huske at rykke den næste linje ind. Thonny hjælper dog med dette, og husker næsten altid at gøre det for dig. Alle de linjer, der er rykket ind efter en linje der afsluttes med et kolon, bliver kun kørt, hvis betingelsen er opfyldt.

Det samme gælder med loops / løkker.

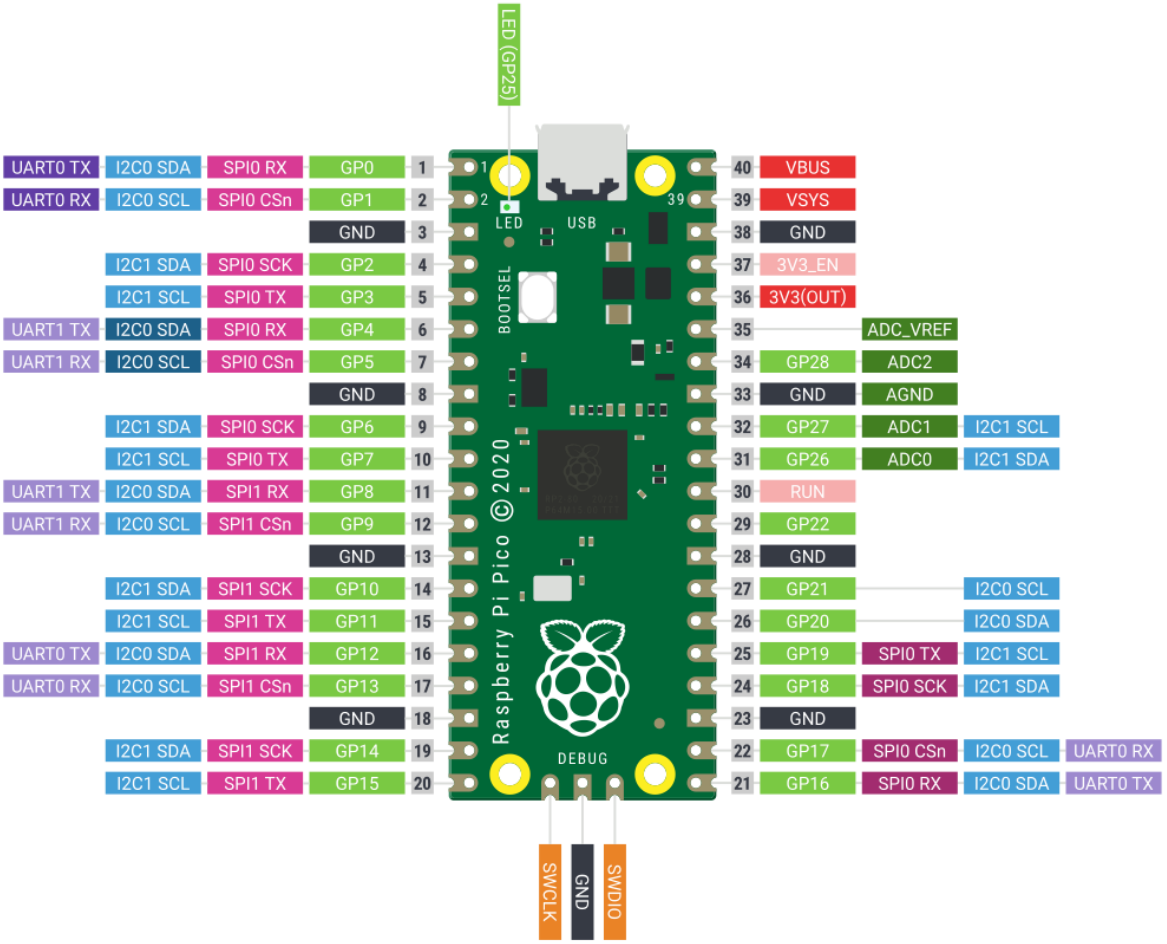
F.eks. `while`, hvor det indrykkede bliver kørt så længe betingelsen er opfyldt:

```
navn = input ("Hvad er dit navn? ")

while navn != "Batman":
    print("Du er ikke Batman - prøv igen!")
    navn = input ("Hvad er dit navn? ")
print("Du er Batman!")
```

Hardware, Raspberry Pi Pico og Python

Picoens pins

For at kunne forbinde picoen med andet hardware, er det vigtigt med en oversigt over dens benforbindelser (pinout) - hvilke der er forsyning (GND, 3,3V eller 5V), samt hvilke der er digitale input eller output eller analoge inputs.



Se den originale PDF fil her:
https://datasheets.raspberrypi.com/pico/Pico-R3-A4-Pinout.pdf

Som man kan se kan de fleste pins eller ben sættes op til at have forskellige funktioner, alt efter hvad man gerne vil bruge dem til. Det er derfor man kalder dem GP (general purpose). Mere om det senere :-)

Lysdiode (Led)

De følgende eksempler viser hvordan man først får den indbyggede lysdiode (led) på picoen til at blinke, og hvordan man efterfølgende kan bygge et kredsløb med en lysdiode og få den til at blinke.

Indbygget Lysdiode

Den indbyggede lysdiode i picoen er, som man kan se af pinout diagrammet ovenover ben GP25 (helt oppe i toppen af figuren).
Programmet for at få led'en til at blinke er sådan her:

```
import machine
import utime

led = machine.Pin(25, machine.Pin.OUT)

while True:
    led.value(1)
    utime.sleep(1)
    led.value(0)
    utime.sleep(1)
```

For at få adgang til alle picoens benforbindelser og forskellige måder at styre dens hardware på skal vi importere biblioteket, machine: `import machine`

Derudover skal vi bruge utime biblioteket, `import utime` - for at kunne holde pauser, når vi blinker med lysdioden - (tænd, pause, sluk, pause).

Vi sætter en variabel, **led**, til at være lig med pin 25 fra vores machine bibliotek (GP25), og sætter samtidig den pin til at være output, da vi jo gerne vil kunne tænde og slukke lysdioden direkte fra Pin'en: `led = machine.Pin(25, machine.Pin.OUT)`

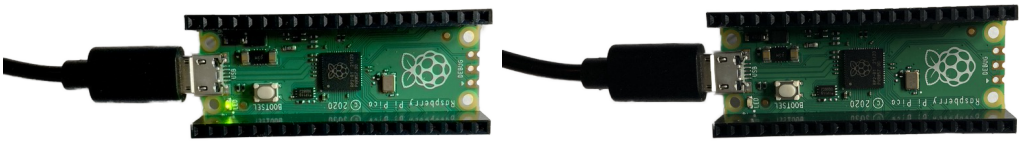
Derefter laver vi et uendeligt loop med `while True:`, hvor vi først tænder led'en ved at skrive: `led.value(1)`

1-tallet betyder at udgangen bliver HIGH - altså 3.3V, og dermed tænder vi lysdioden.

Derefter holder vi en pause i 1 sekund med `utime.sleep(1)`

Og så slukker vi led'en igen med `led.value(0)`

Til sidst holder vi endnu en pause på 1 sekund, for at led'en når at være slukket inden vi tænder den igen, når vi looper tilbage til starten af `while True:` loopet.

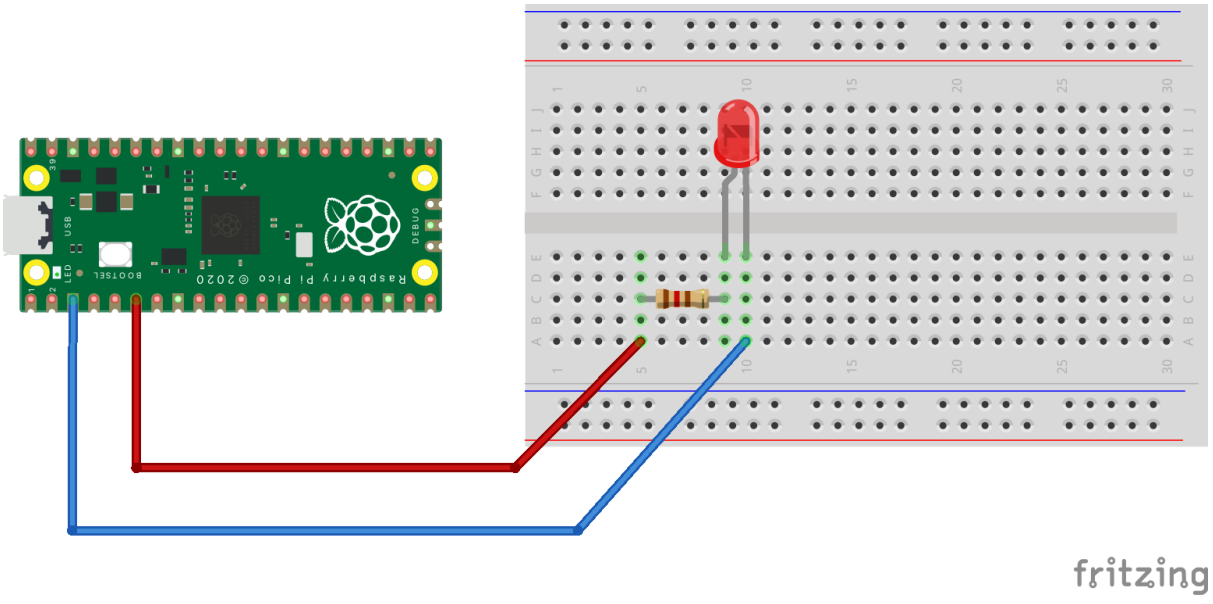


Lysdiode på Breadboard

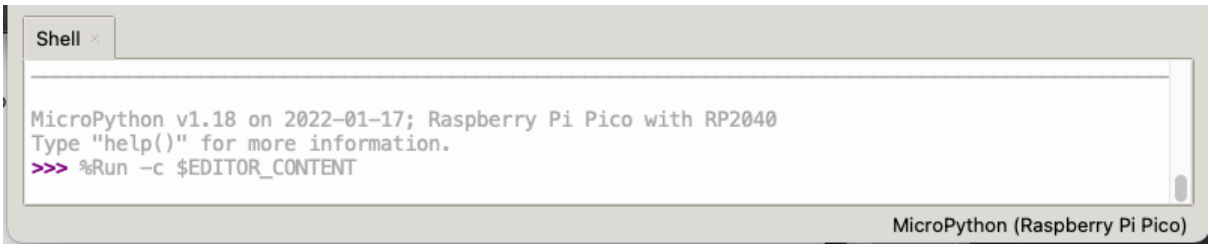
Næste ting vi skal prøve er at bygge en led op på et breadboard og styre den med en anden pin på picoen.

Kredsløbet, der skal bygges er det nedenstående, hvor vi bruger en 120 ohm modstand i seriekredsløb med en lysdiode. Modstanden bruges for at begrænse strømmen gennem og spændingen over lysdioden. Husk at lysdioden skal vendes rigtigt, sådan at det længste ben sidder i samme række som modstanden.

Vi vælger at bruge GP4 (rød ledning) på picoen (ben 6 på venstre side, når vi tæller fra usb-porten) og derudover skal vi bruge GND (Stel eller 0V), som vi kan hente på ben 3.



Prøv at bygge kredsløbet på breadboardet og forbind picoen til usb-porten på din computer. Sørg for at vælge Micropython (Raspberry Pi Pico) i nederste højre hjørne i Thonny:



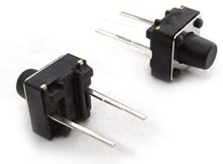
Derefter skal vi bare have ændret programmet ovenstående til at bruge GP4 i stedet for GP25, som vi brugte før til den indbyggede lysdiode.

```
import machine
import utime

led = machine.Pin(4, machine.Pin.OUT)

while True:
    led.value(1)
    utime.sleep(1)
    led.value(0)
    utime.sleep(1)
```

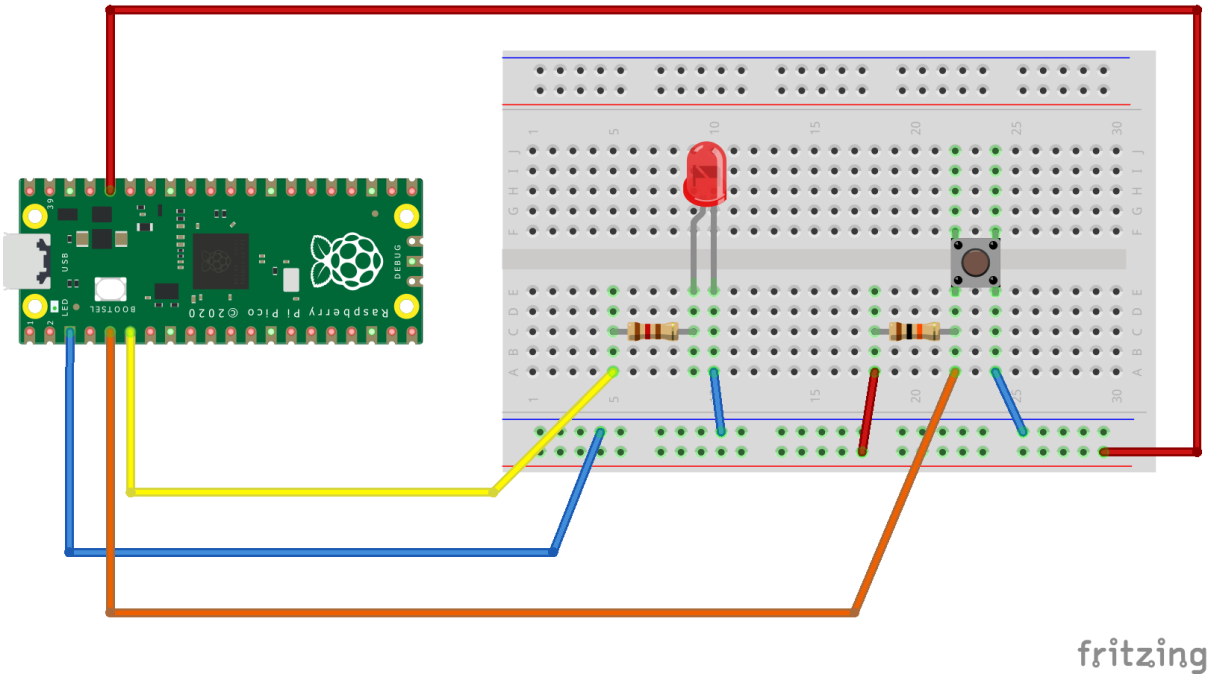
Det er altså kun et enkelt tal der skal ændres fra 25 til 4. Resten af programmet er helt det samme.

Prøv at ændre programmet, så lysdioden blinker hurtigere eller langsommere.

Knap



Hvis man skal læse en knap fra picoen kan man bygge følgende kredsløb. Vi bygger det sammen med lysdiode kredsløbet fra ovenstående, så vi kan bruge knappen til at tænde og slukke led'en.



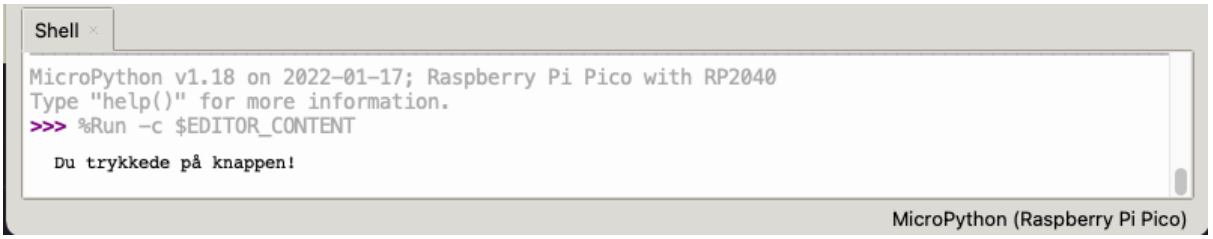
Knappen er forbundet til GP3, og har desuden en pull-up modstand på 10K ohm.

Først checker vi om knappen virker og er forbundet rigtigt med følgende program:

```
import machine
import utime

led = machine.Pin(4, machine.Pin.OUT)
button = machine.Pin(3, machine.Pin.IN)

while True:
    if button.value() == 0:
        print("Du trykkede på knappen!")
        utime.sleep(0.5)
```

Vi har beholdt led-variablen, da vi skal bruge den senere.
Følgende **if** sætning checker om knappen bliver trykket på `if button.value() == 0:`
Da vi i vores kredsløb bruger en pull-up modstand, i stedet for pull-down, vil der være 0V på GP3, når vi trykker knappen ned, og ellers 3,3V. Derfor skal vi checke om knappens værdi er lav / 0, for at checke om den bliver trykket ned.

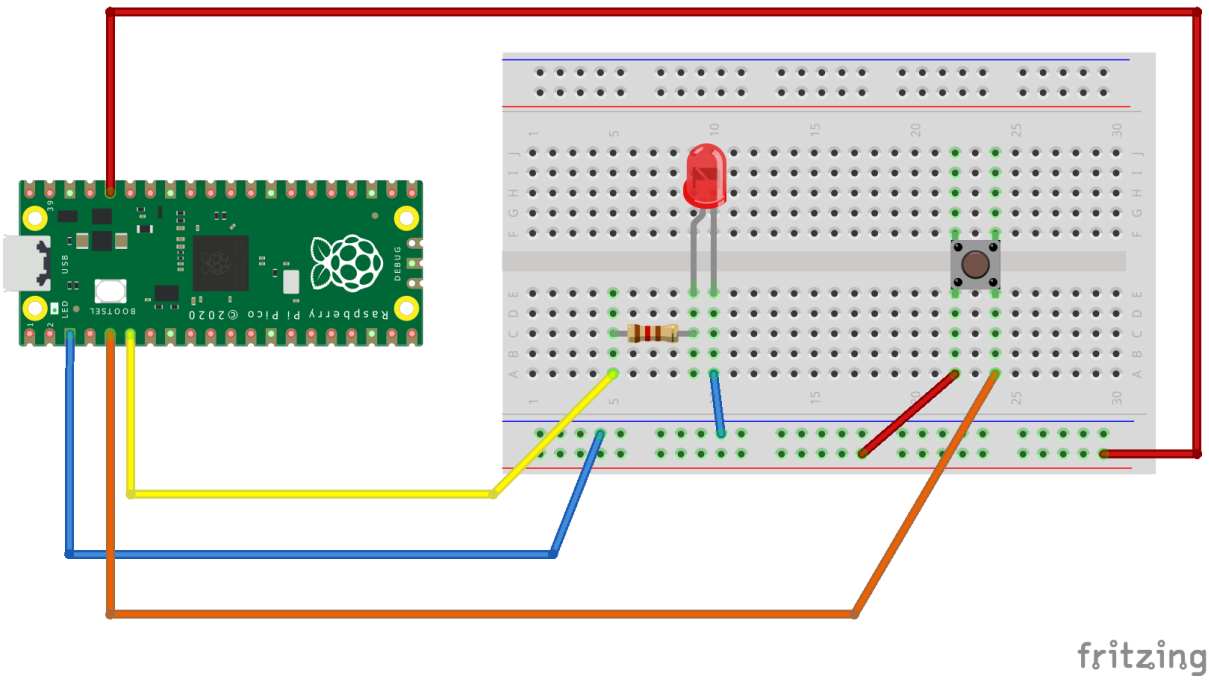
Vi holder også en kort pause med `utime.sleep(0.5)` for at sikre os, at vi ikke skriver ud for ofte (hver halve sekund), da picoen ellers ville skrive “Du trykkede på knappen!” virkelig mange gange, hver gang vi bare trykker kort på knappen.



Virker dit program også? - Og kan du regne ud, hvordan du skal bygge kredsløbet om, hvis du gerne vil kunne skrive `if button.value() == 1:` i stedet for? (Hint der skal bruges en pull-down modstand i stedet for pull-high). Prøv at bygge det og lav ændringen i programmet.

Med picoen er det også muligt helt at undvære pull-up eller pull-down modstande, da den har indbyggede modstande den kan koble til og fra via programmet.

Lad os prøve det:



Bemærk modstanden er fjernet fra knap-kredsløbet, så vi kan bede picoen om at bruge en intern pull-down modstand i stedet. Derfor forsyner vi knappen med 3,3V og fører knappens andet ben tilbage til GP3.

Programmet kommer så til at se således ud:

```
import machine
import utime

led = machine.Pin(4, machine.Pin.OUT)
button = machine.Pin(3, machine.Pin.IN, machine.Pin.PULL_DOWN)
```

```
while True:
    if button.value() == 1:
        print("Du trykkede på knappen!")
        utime.sleep(0.5)
```

Virker det?

Hvis vi nu gerne vil bruge knappen til at tænde og slukke vores eksterne lysdiode med, hvordan skal programmet så se ud?

```
import machine
import utime

led = machine.Pin(4, machine.Pin.OUT)
button = machine.Pin(3, machine.Pin.IN, machine.Pin.PULL_DOWN)

while True:
    if button.value() == 1:
        led.toggle()
        utime.sleep(0.5)
```

Programmet er lavet, så led'en automatisk skifter, hver gang der trykkes på knappen. Hvis den er tændt, slukker den, og hvis den er slukket tænder den. Det kaldes på engelsk for **toggle**, og der findes heldigvis en kommando med samme navn, `led.toggle()` som vi kan bruge.

Prøv at fjerne pausen `utime.sleep(0.5)` fra programmet og se hvad der sker? Toggler den altid til fra tændt til slukket eller slukket til tændt? Hvorfor tror du det virker dårligere uden en pause?

Du kan også prøve at lave et program, der tænder lysdioden, når du trykker på knappen, og automatisk slukker den 5 sekunder senere.

Lyssensor (LDR)

Lyssensoren er ligesom knappen en sensor. Den store forskel med lyssensoren, er at det er en analog sensor, som kan give varieret input til picoen. I lyssensorens tilfælde er input'et afhængigt af styrken af det lys der rammer den. Knappen derimod er en digital sensor, som kun har to forskellige tilstande den kan skiftes imellem, nemlig tændt og slukket, høj eller lav, 0 eller 1.

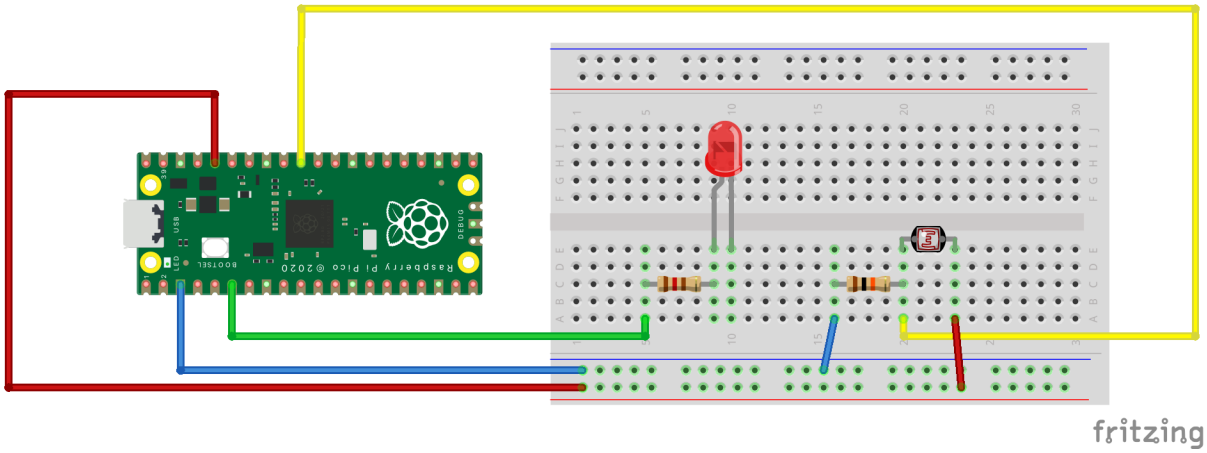
LDR står for Light Dependent Resistor, hvilket på dansk betyder lysafhængig modstand. Det vil sige den ændrer modstand afhængigt af, hvor kraftigt lys der rammer den.

Til at læse analoge signaler (spændinger) har picoen ADC0 (GP26), ADC1 (GP27) og ADC2 (GP28), som konverterer en analog værdi til et digitalt tal (A/D konvertering).
Picoen har en 16bit A/D-konverter. Det vil sige at de spændinger vi giver den (mellem 0 og 3,3V) på enten ADC0, ADC1 eller ADC2 bliver konverteret til et tal mellem 0 og 65.535.

0V = 0 og 3,3V = 65.535

For at man kan omsætte modstandsændringen fra LDR-modstanden til en spændingsændring som kan læses af picoen bruger vi en spændingsdeler, hvor vi sætter LDR-modstanden i serie med en fast modstand på 10K.

Prøv at bygge det her:



Komponenten længst til højre er LDR-modstanden. Vi beholder lysdioden på boardet, så vi kan bruge den igen lige om lidt.

Så skriver vi følgende program:

```
import machine
import utime

ldr = machine.ADC(26)

while True:
    print(ldr.read_u16())
    utime.sleep(1)
```

Vi sætter en ny variabel, LDR til at være et ADC input på GP26 - altså ADC0 med `ldr = machine.ADC(26)`

Derefter printer vi i et loop den konverterede værdi vi læser fra LDR-sensoren med `print(ldr.read_u16())`, hvor u16 fortæller picoen, at det er en 16-bit værdi vi læser.

Derefter holder vi en pause på 1 sekund med `utime.sleep(1)` inden loopet starter forfra.

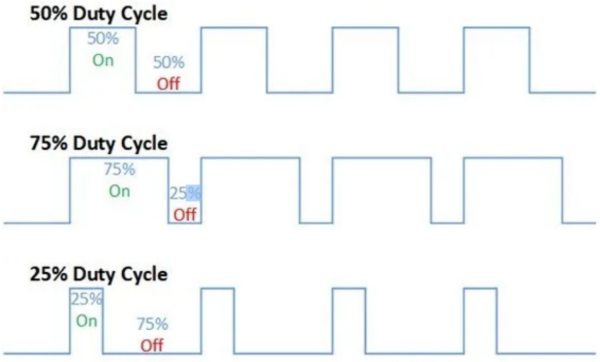
- Prøv at dække for LDR-sensoren, så det bliver helt mørkt. Hvilken værdi kan du få den ned på i Thonny?
- Prøv også at lyse på LDR-modstanden. Hvilken værdi kan du få den op på?

Nu kan vi prøve at styre LED'en med LDR-sensoren, sådan at jo mørkere det bliver, jo mere skruer vi ned for LED'en.

Prøv med følgende program:

```
import machine
import utime

ldr = machine.ADC(26)
led = machine.PWM(machine.Pin(4))
led.freq(1000)

while True:
    ldr_value = ldr.read_u16()
    print(ldr_value)
    led.duty_u16(ldr_value)
    utime.sleep(0.25)
```

Først sætter vi ligesom før en variabel op til ldr, og derefter sætter vi en variabel op til led'en med `led = machine.PWM(machine.Pin(4))`. PWM funktionen betyder at vi sætter GP4 Pin'en op til at kunne bruge Pulsbreddemodulation (Pulse Width Modulation eller forkortet PWM) til at skrue op og ned for lysstyrken på led'en. I praksis tænder og slukker man bare for led'en så hurtigt at øjet ikke kan opfange det - `led.freq(1000)` - nemlig 1000 gange i sekundet, og så justerer man på hvor lang tid led'en er tændt ift hvor lang tid den er slukket (Duty Cycle). Når led'en har en Duty Cycle på 50% er den tændt og slukket lige lang tid, og derfor ser det ud som om den lyser med halv styrke. Hvis Duty Cycle er 75% er led'en tændt 75% af tiden og slukket de sidste 25% - så ser det ud som om den lyser ved næsten fuld styrke.



Hvis man nu skulle have led'en til at lyse kraftigere og kraftigere jo mørkere det bliver, hvad skal man så gøre?

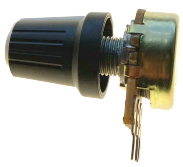
Prøv f.eks. dette:

```
import machine
import utime

ldr = machine.ADC(26)
led = machine.PWM(machine.Pin(4))
led.freq(1000)

while True:
    ldr_value = ldr.read_u16()
    #print(ldr_value)
    led.duty_u16(65535-ldr_value)
    utime.sleep(0.02)
```

Her har vi trukket ldr_value fra den størst mulige værdi den kan have, nemlig 65535 - `led.duty_u16(65535-ldr_value)` for at få den til skrue ned for lyset, jo kraftigere lyset er på LDR-sensoren.

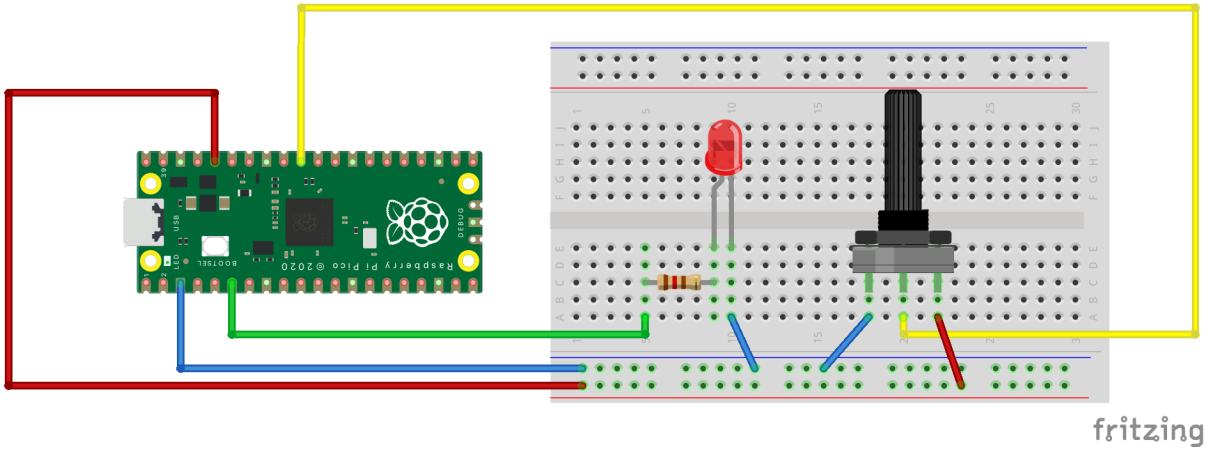
Derudover har vi sat pausen ned - `utime.sleep(0.02)` og kommenteret print-linjen ud med # - `#print(ldr_value)`. # bruges i Python til at fortælle at en linje ikke skal tages med, når programmet kører. Man kan både bruge det til at lave små stykker tekst der beskriver ens kode, eller man kan bruge det til at slå programlinjer fra, så de ikke tages med, når programmet kører.

Potentiometer



Et potentiometer er ligesom LDR-sensoren en variabel modstand, der ændrer modstand alt efter hvor meget den er drejet. I virkeligheden fungerer et potentiometer som en

spændingsdeler, hvor spændingen deles mellem de 2 variende modstande, der er mellem det første og det midterste ben og det midterste og det sidste ben på potentiometeret.

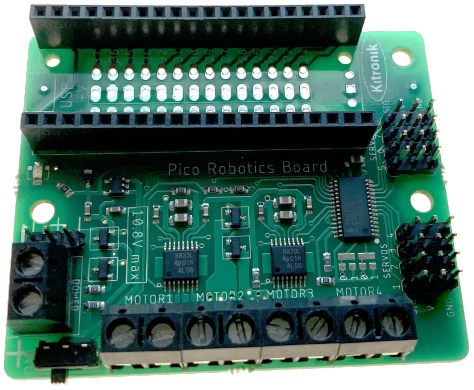


For at læse potentiometeret kan vi bruge næsten det samme program som til LDR-sensoren:

```
import machine
import utime

potmeter = machine.ADC(26)
led = machine.PWM(machine.Pin(4))
led.freq(1000)

while True:
    potmeter_value = potmeter.read_u16()
    print(potmeter_value)
    led.duty_u16(potmeter_value)
    utime.sleep(0.25)
```

Prøv i stedet at lave et program der ændrer på hvor hurtigt led'en blinker, når man drejer på potentiometeret.

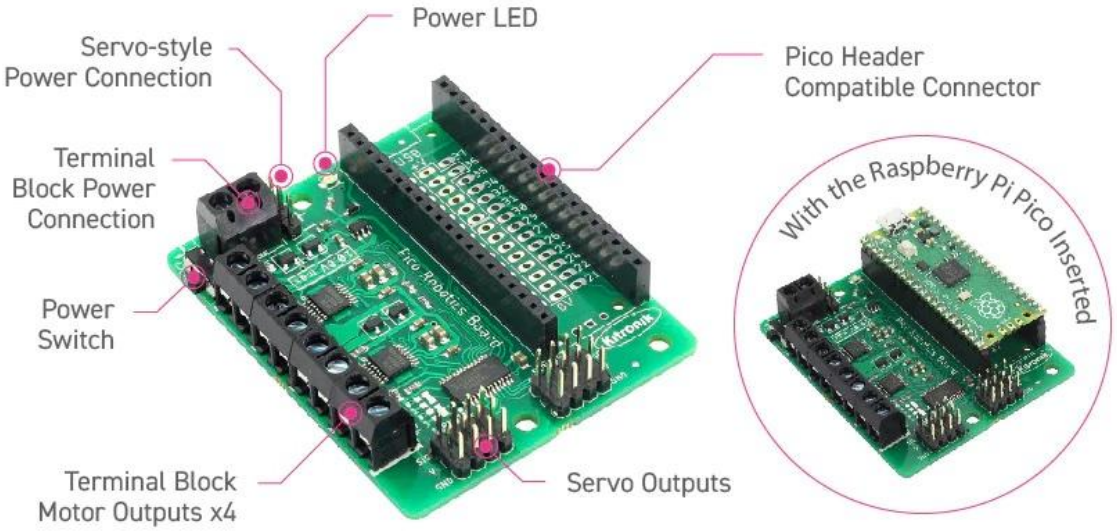
Pico Robotics Board - fra Kitronik



https://kitronik.co.uk/products/5329-kitronik-compact-robotics-board-for-raspberry-pi-pico

Robotics boardet er beregnet til at styre motorer med. Man kan både styre almindelige DC-motorer, og servo-motorer. Derudover kan man tilslutte en batteriforsyning til den, som kan bruges til at forsyne både pico'en og de motorer man tilslutter.

Boardet er bygget, så man kan montere pico'en direkte ned i det. Se på nedenstående figur, hvordan du skal vende den.



Når man skal bruge Robotics boardet fra Kitronik skal man først have installeret et bibliotek til alle de kommandoer man skal bruge i Python. Der er en guide til sidst i dokumentet til hvordan man gør det.

Når det er installeret kan man i Thonny skrive:

```
import PicoRobotics
```

Og så er vi klar til at prøve med nogle motorer.

DC Motor

En DC Motor er en motor man kan styre med jævnspænding - altså den type spænding der kommer fra f.eks. batterier. Motorens hastighed styres med spænding - altså lav spænding = lav hastighed og høj spænding = høj hastighed. Dog er der grænser både for hvor lav spændingen må være for motoren overhovedet kan køre og hvor høj spændingen må være for at motoren ikke går i stykker.

Den gule motor vi bruger er beregnet til en spænding på 3-6 Volt. Den kan dog godt tåle lidt højere spænding end 6V, men den vil ikke køre hvis spændingen er lavere end 3V.

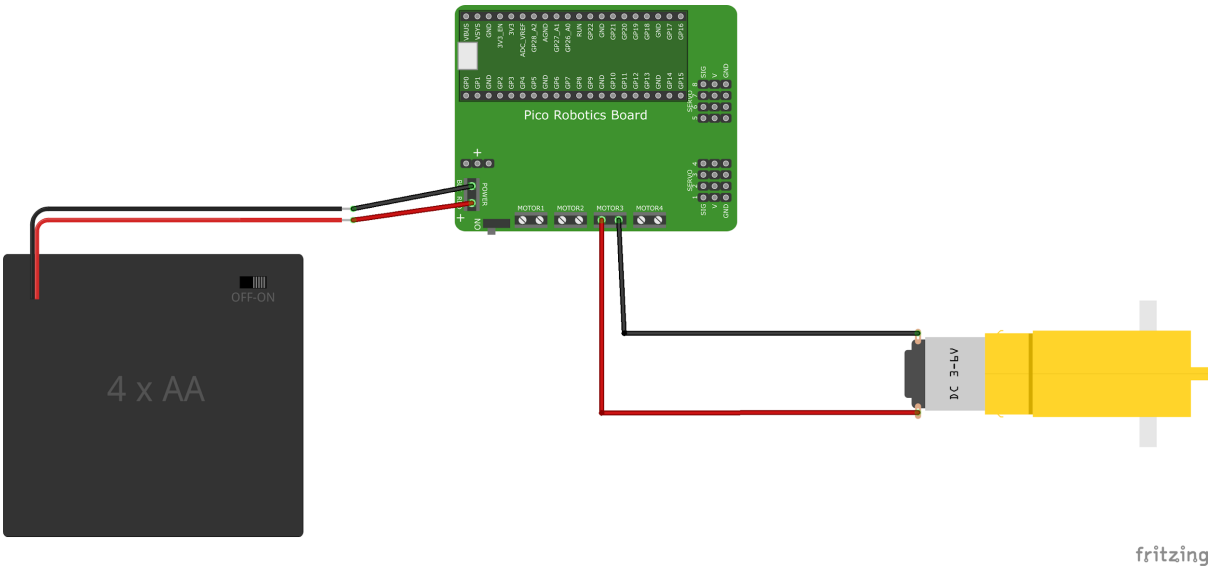
Med Pico Robotics boardet styrer vi dog motoren med et tal mellem 0 og 100 for at angive hvor hurtigt motoren skal køre.

En DC motor kan man kun styre hastigheden på, man ved ikke noget om, hvor mange omgange eller grader den er roteret.

Vores motor har også en indbygget gearkasse på enten 120:1 eller 45:1, der gør at den kører langsommere og trækker større kraft, end hvis den ikke havde gearing.

Hvad tror du 120:1 eller 45:1 betyder?

Prøv at bygge dette:



Husk at sætte batterier i, og tænde for batterikassen og for power på robotics board'et.
Du skal bruge en lille skruetrækker for at sætte ledningerne fast i robotics boardet.

Skriv følgende program i Thonny og prøv det.

```
import PicoRobotics
import utime

board = PicoRobotics.KitronikPicoRobotics()

while True:
    board.motorOn(3,"r",100)
    utime.sleep_ms(1000)
    board.motorOff(3)
    utime.sleep_ms(1000)
```

Motorkommandoen `board.motorOn(3,"r",100)` tænder for motor 3, og sætter den til at køre baglæns - "r" (reverse), og kører med max hastighed på 100.

Hvis man vil have motoren til at køre fremad skal man bruge "f" (forwards) i stedet.

Prøv at få motoren til at køre fremad med hastighed 25.

Prøv at få motoren til at skifte mellem at køre fremad og baglæns

Prøv at få motoren til langsomt at skrue op for hastigheden fra 0 til 100.
HINT - man kan bruge en for-løkke eller en while-løkke, hvor man hver gang løkken starter forfra tæller en variabel op med 1, og bruger den variabel til at styre hastigheden på motoren.

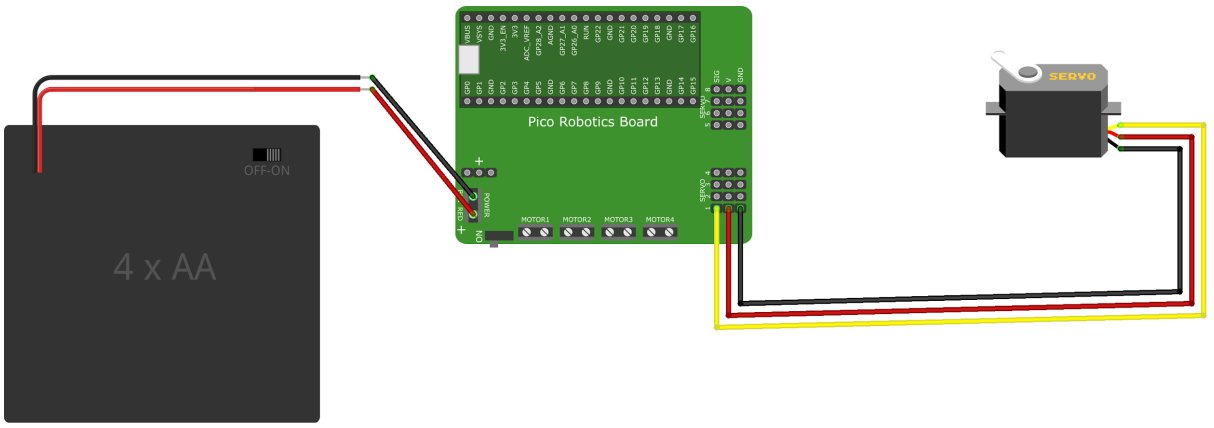
Servo

En Servomotor er en motor, der kan styres til at stille sig i en bestemt position - rotere et bestemt antal grader ift. dens udgangspunkt - 0 grader. De fleste servoer kan ikke rotere mere end 180 grader - ligesom den vi har i vores sæt.

De bruges ofte til at styre simple men præcise bevægelser, som f.eks. haleroret på et modelfly eller en simpel robotarm, eller døråbnere m.m.

Pico robotics board'et har indbygget styring af op til 8 servomotorer.

Prøv at bygge følgende (husk at vende servostikket rigtigt så det passer med farverne på ledningerne):



Først laver vi et simpelt styringsprogram, der først sætter servo 1 til 0 grader, derefter 90 grader, og 180 grader og tilbage til 90, og så forfra…

```
import PicoRobotics
import utime

board = PicoRobotics.KitronikPicoRobotics()

while True:
    board.servoWrite(1, 0)
    utime.sleep_ms(500)
    board.servoWrite(1, 90)
    utime.sleep_ms(500)
    board.servoWrite(1, 180)
    utime.sleep_ms(500)
    board.servoWrite(1, 90)
    utime.sleep_ms(500)
```

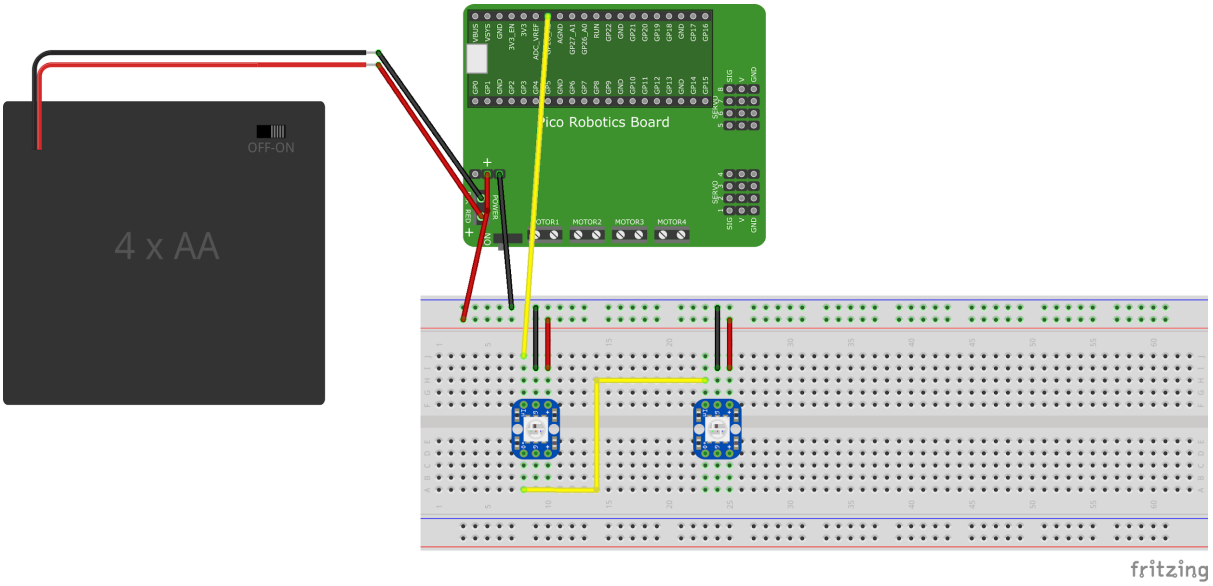
Hvordan vil du få servoen til at bevæge sig mere flydende mellem 0 og 180 grader?
Prøv om du kan skrive et program der får den til at gøre det - f.eks. med en while-løkke eller en for-løkke.

NeoPixel

En Neopixel er en lysdiode, der kan lyse i alle regnbuens farver. Neopixels er lette at sætte sammen i en kæde - man forbinder bare output på den ene med input på den næste, o.s.v., og hele kæden kan styres med kun en enkelt output pin på pico'en.

For let at kunne styre neopixels fra Pyton skal du følge guiden til at installere biblioteket til den til sidst i dette dokument.

Når du har installeret det kan vi gå igang med at bygge:



Vi forbinder i dette tilfælde in på den første neopixel med GP28 på pico'en.

Vi tager Spændingen (+ og -) fra batteriforsyningen til Pico Robotics Board'et

Nu kan vi prøve at skrive et program:

```
import time
from neopixel import Neopixel

pixels = Neopixel(2, 0, 28, "GRB")

yellow = (255, 100, 0)
orange = (255, 50, 0)
green = (0, 255, 0)
blue = (0, 0, 255)
red = (255, 0, 0)

while True:
    pixels.brightness(50)
    pixels.fill(orange)
```

```
    pixels.show()
    time.sleep(1)
    pixels.fill(blue)
    pixels.show()
    time.sleep(1)
```

I linjen `pixels = Neopixel(2, 0, 28, "GRB")` fortæller vi at vi gerne vil lave en kæde med 2 neopixels. 0'et er ikke vigtigt, og derefter fortæller vi den at vi vil styre kæden med GP28 benet på pico'en, og at rækkefølgen af farverne for den type neopixel vi bruger er Grøn, Rød, Blå - "GRB".

I de næste linjer definerer vi nogle variabler til at holde nogle farver, hvor vi sætter mængden af hhv rød, grøn og blå, der skal blandes. Værdien for hver grundfarve skal være mellem 0 og 255 - hvor 255 er helt rød, grøn eller blå.

Derefter sætter vi lysstyrken for alle pixels til 50% med linjen `pixels.brightness(50)`

Så bruger vi kommandoen fill, til at give samme farve til alle neopixels i kæden - `pixels.fill(orange)`

For at få vist de farver og den lysstyrke vi har sat for vores neopixels er det nødvendig at bruge kommandoen show() - `pixels.show()`

Det skal man altså gøre hver gang man har lavet ændringer i farverne for ens neopixelkæde.

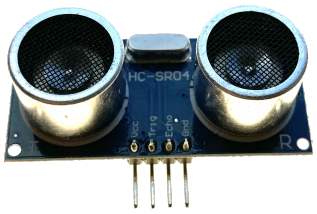
Der findes også kommandoer til at sætte farven for en enkelt neopixel i en kæde:

```
pixels.set_pixel(2, (255, 255, 255))
pixels.show()
```

Her sætter vi direkte farven for pixel 2 i kæden til farven hvid (255, 255, 255).

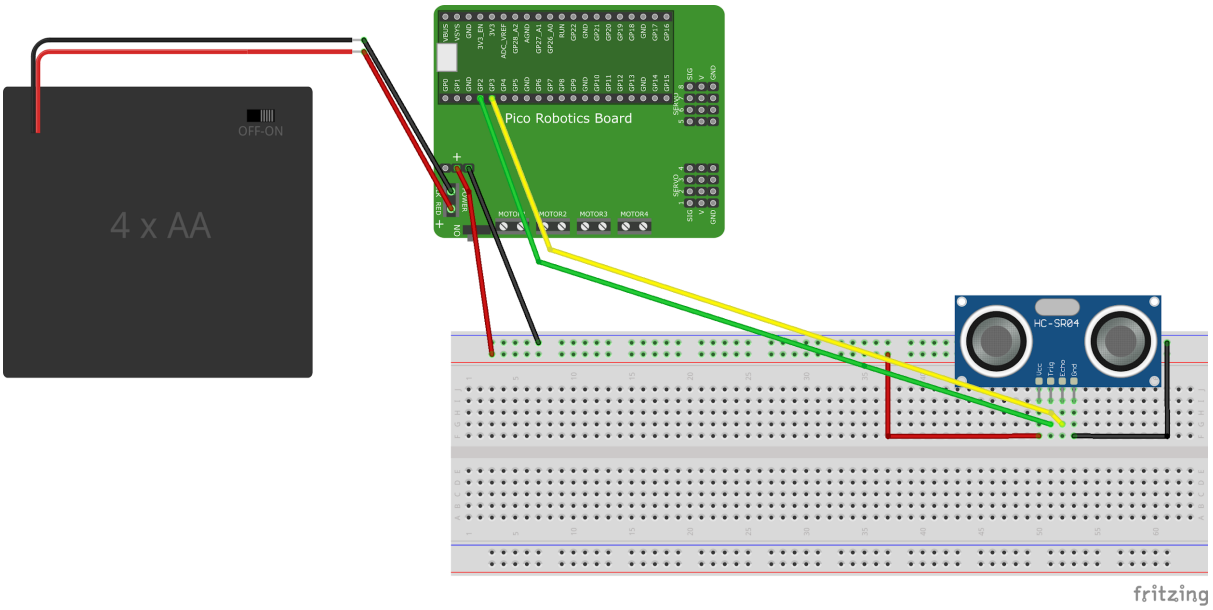
Prøv at lave et program, hvor dine 2 neopixels skifter mellem farverne gul og grøn, så den ene er gul mens den anden er grøn og omvendt.

Sonarsensor



Sonarsensoren er en afstandssensor, der fungerer lidt som en flagermus. Den sender ultralyd (lyd man ikke kan høre) ud og måler den tid, der går indtil lyden vender tilbage - f.eks. hvis lyden rammer et eller andet, hvor den kastes tilbage igen. det kan være en væg eller en bold eller… Når man kender den tid det har taget lyden at bevæge sig frem og tilbage kan man bruge lydens hastighed til at beregne, hvor langt den har bevæget sig.

Prøv at bygge følgende:



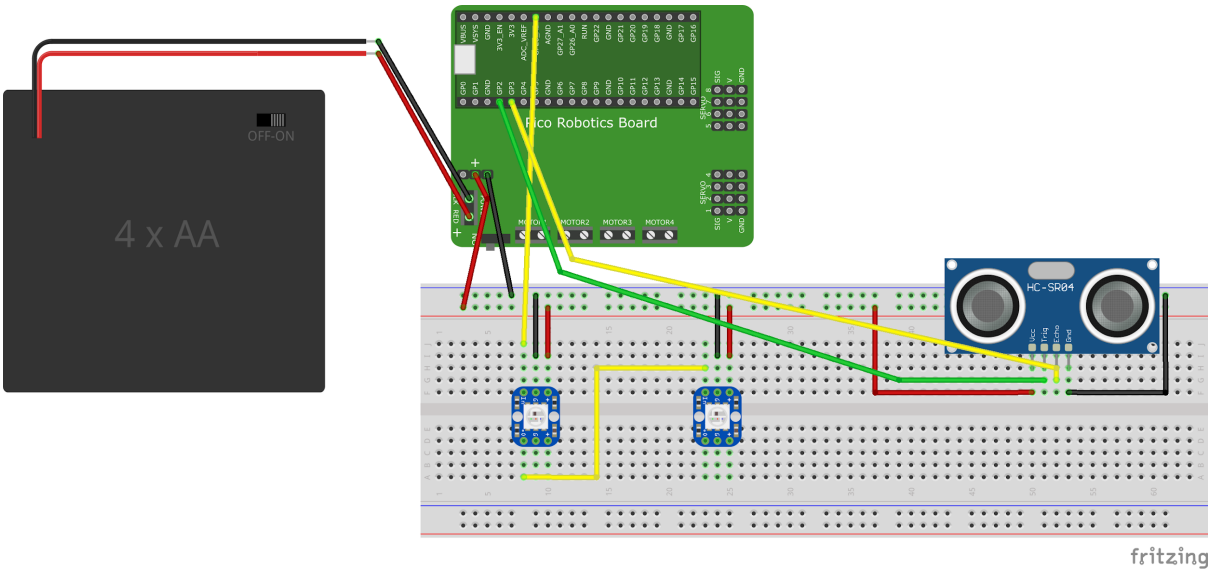
Sonaren har udover spændingsforsyningen fra Pico Robotics Boardet også en trigger og en echo pin. Triggeren er den der sender sonar-lyden afsted, og echo er den pin, der går høj, når der modtages et ekko.

For at bruge sonaren er det lettest at bruge det bibliotek, der findes til micropython. Der er link til det nederst i dokumentet. Det installeres på pico'en ligesom neopixels og Pico Robotics Board bibliotekerne.
Prøv at lave følgende program:

```
import time
from hcsr04 import HCSR04

sensor = HCSR04(trigger_pin=2, echo_pin=3)


while True:
    distance = sensor.distance_cm()
    print('Distance:', distance, 'cm')
    time.sleep(1)
```

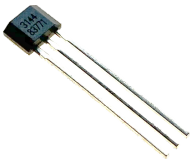
Afstanden skrives hvert sekund i prompten i shell'en i Thonny, så snart programmet kører.

Prøv at bygge følgende:

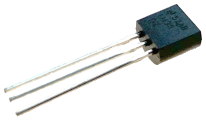


Og skriv så et program, der lyser rødt hvis afstanden til et objekt er mindre end 10 cm, gult, hvis afstanden er mellem 10 og 20 cm og grønt, hvis afstanden til et objektet er mere end 20 cm.

Hall sensor (Magnetsensor)



Temperatursensor

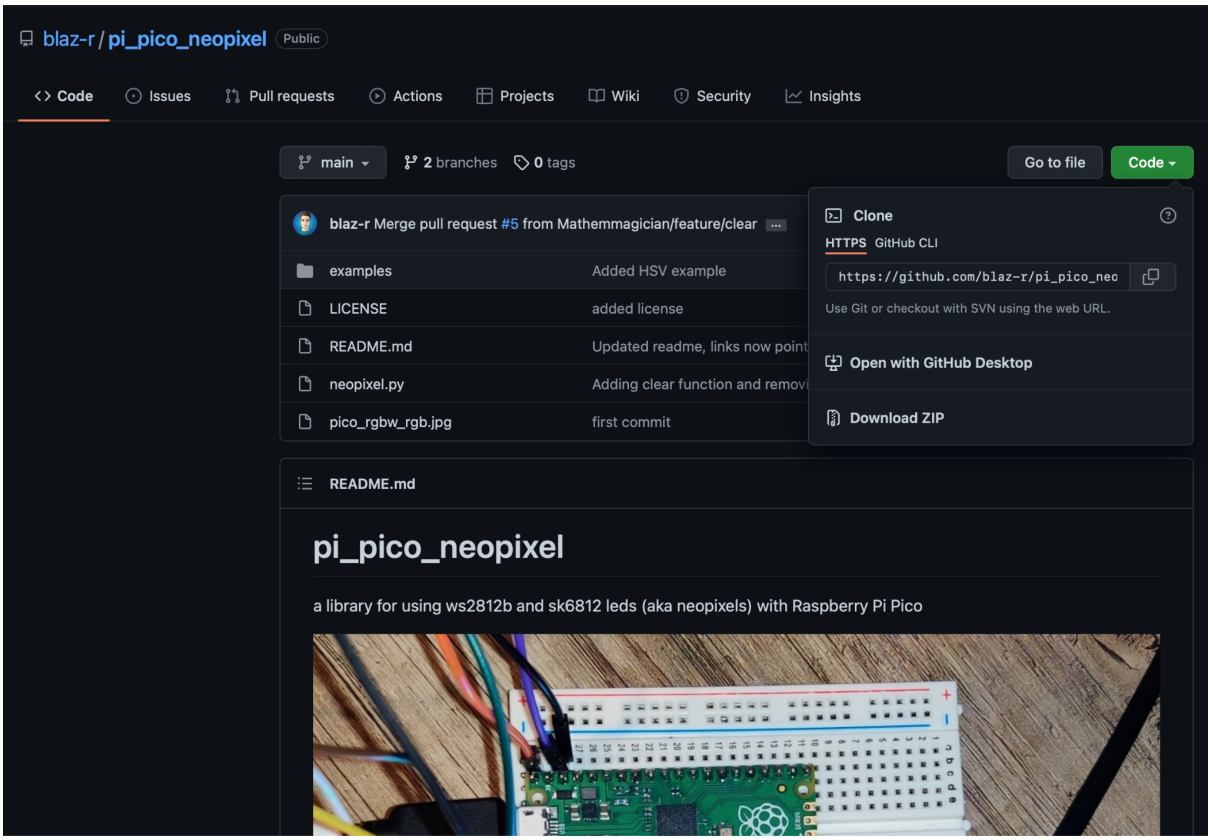


Linjesensor

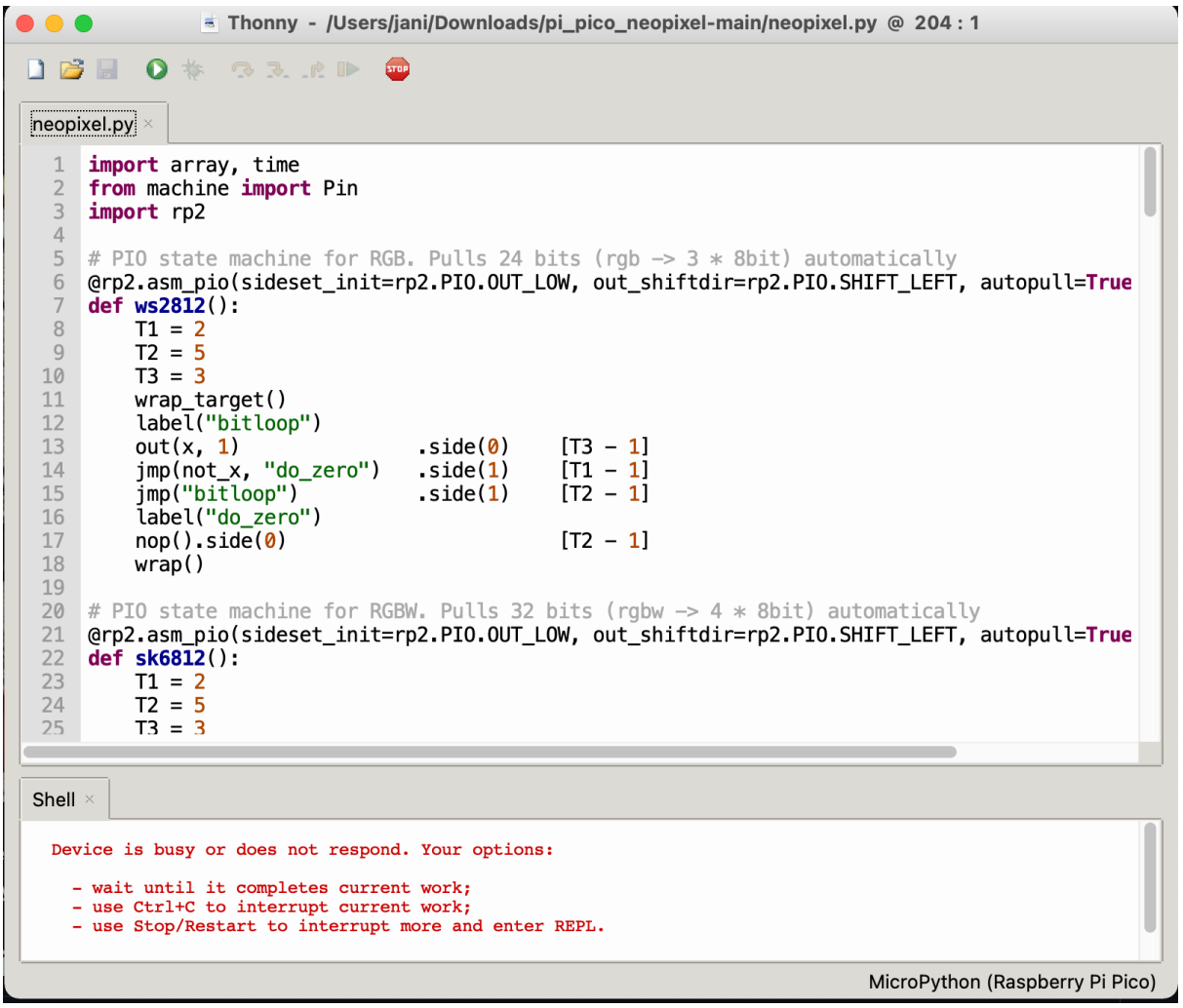
eller

Python biblioteker til Pico - fx Neopixel og Robotics board'et fra Kitronik

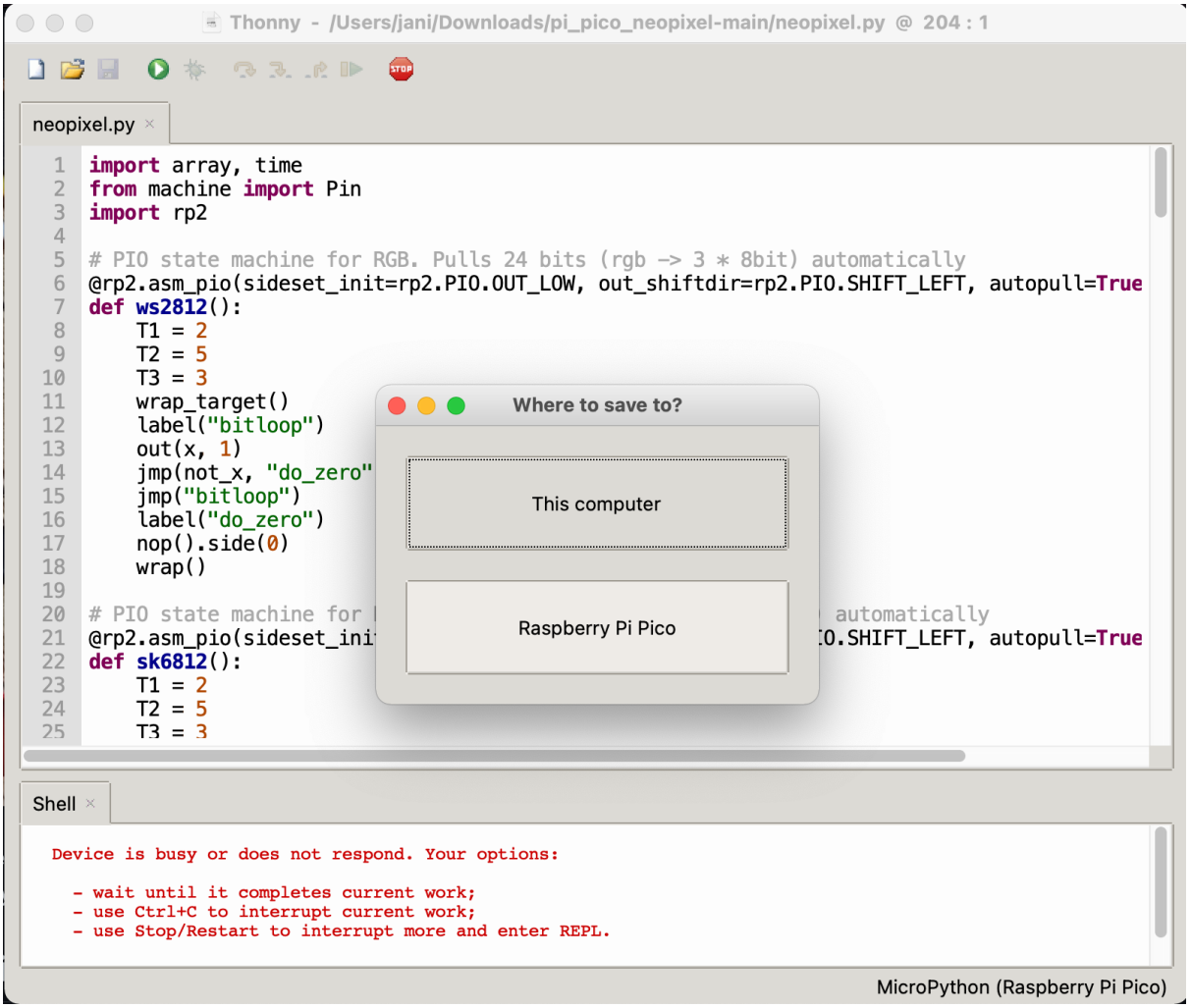
For at bruge nedenstående biblioteker skal de hentes fra nedenstående links - Tryk på Code-knappen og vælg Download ZIP som vist i nedenstående billede.



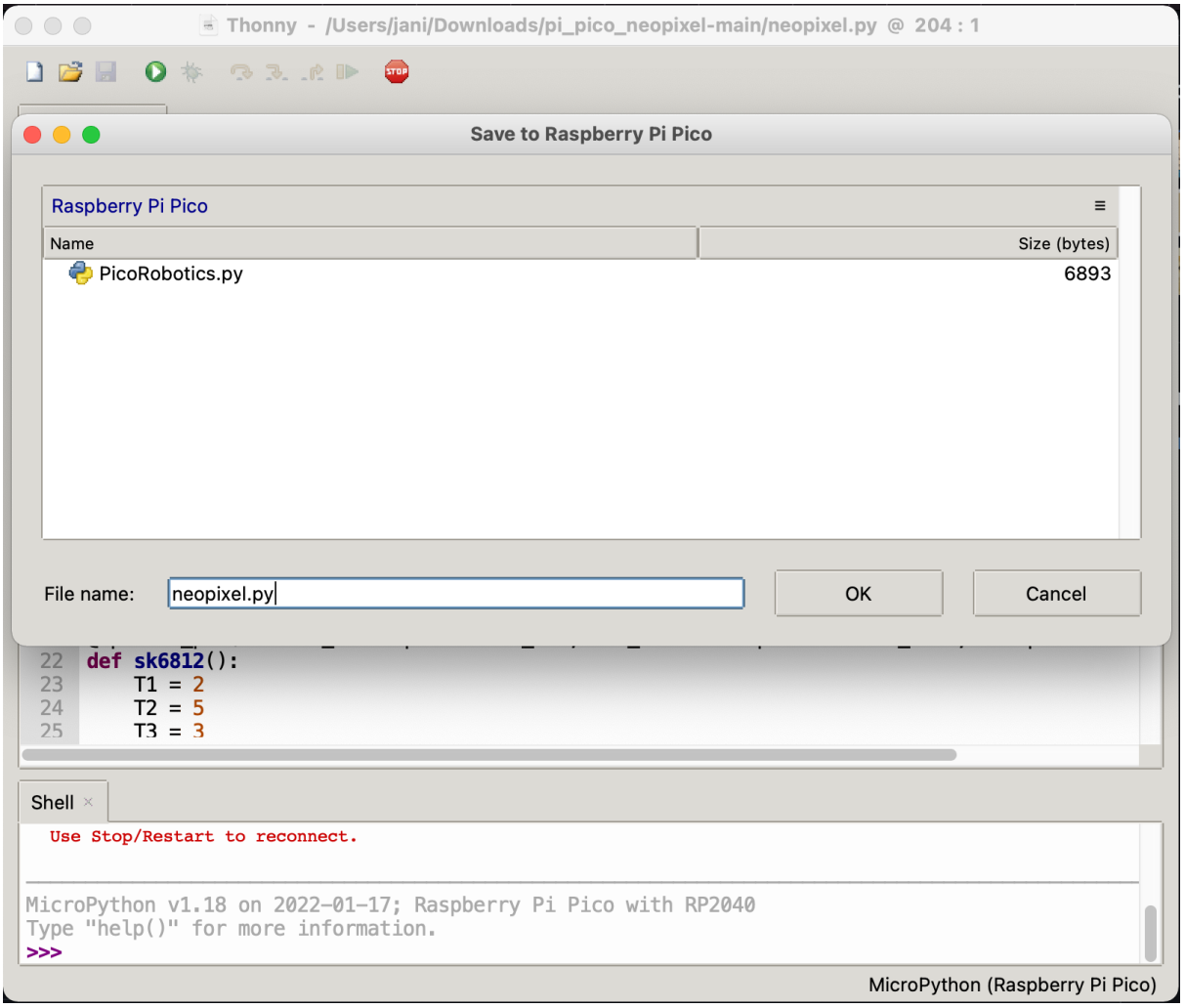
Når zip-filen er hentet ned på din PC, pakker du den ud og finder den fil, der hedder det samme som biblioteket, f.eks. neopixel.py og åbner den i Thonny.



I Thonny kan du så få den fil overført til picoen sådan her, så du kan kalde den fra de programmer du laver.



Og giv den samme navn som den fil du åbnede



Link til bibliotekerne:

Pico Robotics Board fra Kitronik:
https://github.com/KitronikLtd/Kitronik-Pico-Robotics-Board-MicroPython

Neopixel biliotek:
https://github.com/blaz-r/pi_pico_neopixel

Sonar bibliotek (HC-SR04):
https://github.com/rsc1975/micropython-hcsr04

https://www.freva.com/transfer-files-between-computer-and-raspberry-pi-pico/