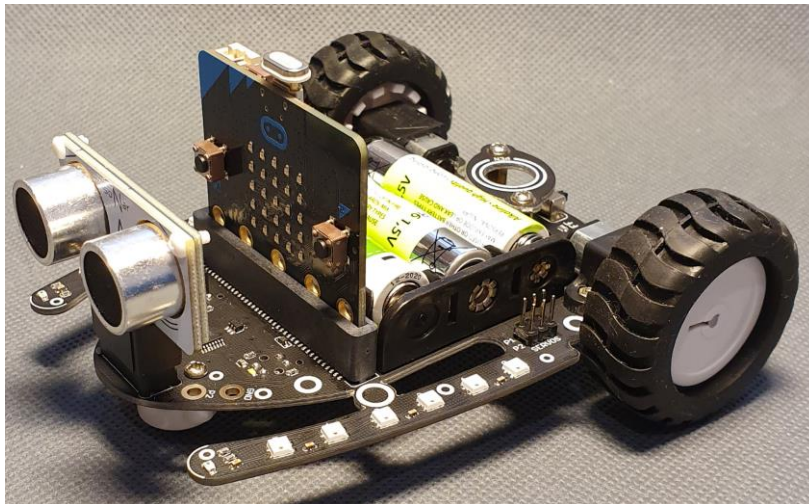


# Python med micro:bit (browser)

Af: Michael Hansen, version 1.90

Dokumentet ligger på: <https://github.com/mhfalken/microbit/>



Formålet er at lære simpel programmering i Python, hvor man bruger en micro:bit og en Bit:Bot bil som platform. Dokumentet bruger en browser baseret Python editor, som ikke kræver nogen installation og som har et interface som ligner blokprogrammering.

Selve dokumentet er baseret på 'learning by doing', dvs der er ikke meget Python undervisning i selve dokumentet.

Hovedsporet i dokumentet er rimeligt let og vel forklaret, mens ekstraopgaverne er noget sværere. Der er løsninger til de fleste opgaver i slutningen af dokumentet. Dokumentet indeholder også en kort Python reference, selvom min erfaring er, at den gider børnene ikke at læse alligevel.

Til at gøre programmeringen af bilen lettere, har jeg lavet et Bit:Bot modul, som ligger ved siden af dokumentet (se specifik link i toppen af dokumentet).

Målgruppen er børn fra 12-årsalderen som ønsker at lære Python ved brug af en micro:bit og en bil. De første dele af dokumentet kan laves uden brug af Bit:Bot bilen.

1	Introduktion.....	3
1.1	Python i browser .....	3
1.2	Hello world .....	3
2	Micro:bit .....	5
2.1	Lysdioderne .....	5
2.2	Lav dit eget billede.....	5
2.3	Brug knapperne A og B .....	5
2.4	Lav en terning .....	6
2.5	Reaktionstest .....	6
2.6	Vaterpas.....	7
2.7	Skridttæller .....	7
2.8	Kommunikation .....	8
3	Hardware .....	9
3.1	Styr en LED.....	9
3.2	Styr nogle neopixels.....	9
3.3	Alarm hvis den flyttes .....	10
3.4	Tilslut andre enheder .....	10
4	Bit:Bot bil .....	11
4.1	Bit:Bot Python modul .....	11
4.2	NeoPixels .....	11
4.3	Firkant.....	12
4.4	Følg linje.....	12
4.5	Sonar.....	13
5	Appendiks – ekstra opgaver .....	14
5.1	Bit:Bot bil .....	14
5.2	Ideer til videre forløb.....	14
6	Appendiks – info .....	16
6.1	bitbot.py .....	16
6.2	Løsninger .....	16
7	Python reference .....	25
7.1	Generelle .....	25
7.2	Datatyper.....	25
7.3	Kontrol .....	26
7.4	Loops.....	27
7.5	Funktioner .....	27

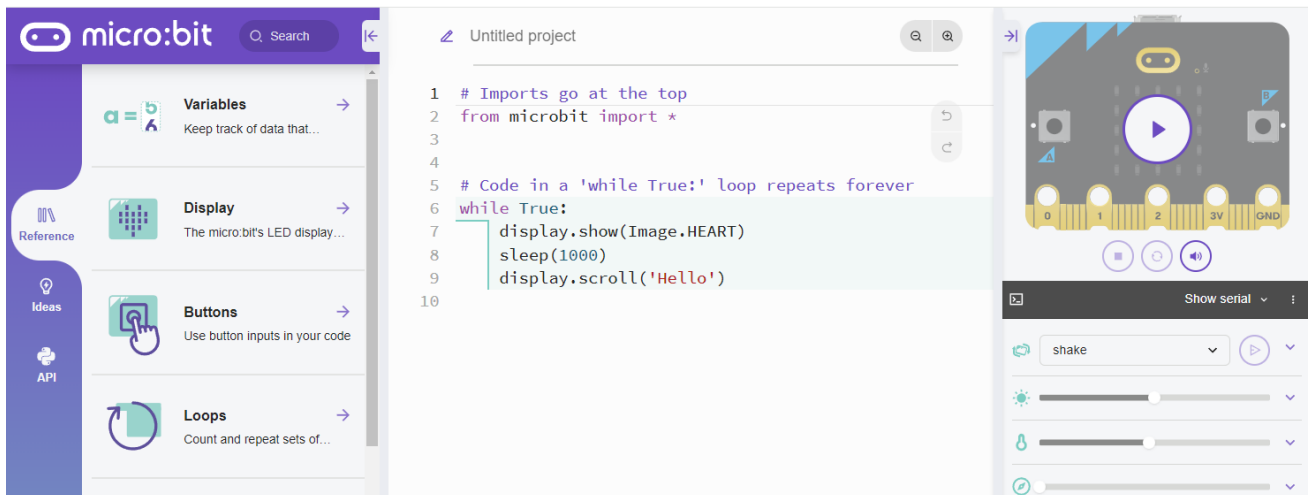
# 1 Introduktion

## 1.1 Python i browser

Det kræver ingen installation eller registrering at bruge denne Python version da Python køres direkte i en browser ved at trykke på følgende link:

<https://python.microbit.org/v/3>

(Det kan være en fordel at bruge Chrome, men det burde virke i andre browsere også).

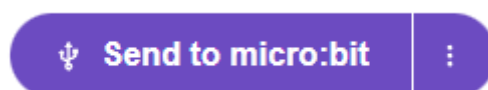


Ude til venstre kan man finde hjælp og eksempler på Python kode som kan trækkes direkte ind kodevinduet i midten. Ude til højre er der en simulator, hvor man kan køre sin kode uden at have en micro:bit. Man kan også bare taste koden ind direkte og så hjælper editoren med at få styr på syntaksen.

## 1.2 Hello world

Når man starter editoren op, så kommer der automatisk et 'Hello world' eksempel frem.

Tilslut nu en micro:bit til computeren og tryk på **Send til micro:bit** i bunden af skærmen for at downloade det til micro:bit'en.



Følg dialogen for at få kontakt til micro:bit'en inden koden downloades. Første gang man downloader noget kode til micro:bit'en så tager det noget tid, da hele Python systemet skal downloades først. Det er kun første gang at det sker og derefter går det meget hurtigere.

- 1) Se at koden virker som den skal.

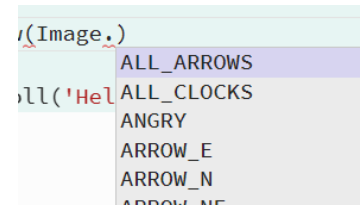
Kort gennemgang af hvad koden laver:

- |   |   |
|---|---|
| • <code>from microbit import *</code>     | denne linje skal altid være der i alle programmer man laver |
| • <code>while True</code>                 | loop for evigt  |
| • Indrykning                              | Rigtig indrykning er vigtig i Python, brug TAB knappen      |
| • <code>display.show (Image.HEART)</code> | Viser et grafikbillede på de røde lysdioder                 |
| • <code>sleep(1000)</code>                | Venter i 1000 millisekunder (= 1 sekund)                    |
| • <code>display.scroll ('Hello')</code>   | Viser tekst på de røde lysdioder                            |

Hvis man vil gemme filen så skal man trykke på **Save** nede i bunden af vinduet og give det et navn. Den gemmer en hex fil (og ikke en Python fil) som kan bruges til at gendanne ens fil eller projekt med. Det er en god ide at gemme ens projekt når man er færdig med det, så man kan komme tilbage til det senere.

## Opgave

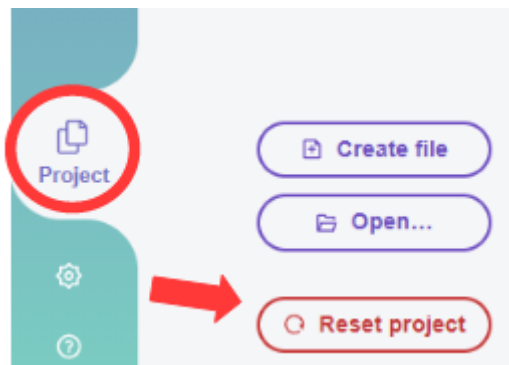
- 1) Vælg et andet grafikbillede. Hint: Editoren kan hjælpe her: Slet `.HEART` og sæt derefter et nyt `.` lige efter `Image`. Nu viser editoren en liste over alt det man kan skrive her.
- 2) Ret teksten 'Hello' til noget andet
- 3) Gem programmet som *hello-world*.



## 2 Micro:bit

Man kan styre lysdioderne på flere forskellige måder.

Start med at lave et nyt projekt. Det gør man ved at trykke på **Project** nede i venstre hjørne og vælge **Reset project**.

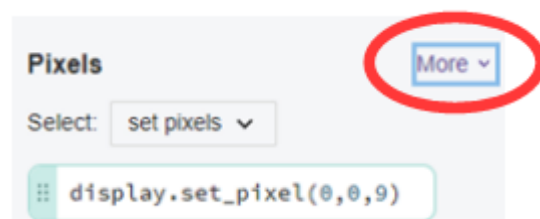


I **Referencen** hedder lysdioderne for **Display**. Tryk ude til venstre på **Reference** og derefter på **Display**.

### 2.1 Lysdioderne

Vi har allerede brugt de første metoder, så prøv at tænde en lysdiode ved at bruge `display.set_pixel()` som står nederst.

Hvis man har brug for mere hjælp, så tryk på **More** oppe i højre hjørne.



#### Opgave

- 1) Tænd en af lysdioderne
- 2) Ændre lysstyrken på lysdioden
- 3) Lav en loop som tænder flere lysdioder (hint: tryk på **More**)
- 4) Ændre loop'en så lysdioderne har forskellig lysstyrke (hint: brug `x` eller `y` fra loop'en)
- 5) Lav to loops som tænder alle lysdioderne
- 6) Hvis eksemplet fra **More** er brugt, så prøv at ændre `9` til `x+y`

Gem programmet som *led-fill*.

### 2.2 Lav dit eget billede

I stedet for at styre lysdioderne en af gangen, kan man lave sine egne grafikbilleder. Det gøres med `display.show(Image(...))` se **Referencen**.

#### Opgave

- 1) Lav nyt projekt
- 2) Lav dit eget billede vha `Image` funktionen
- 3) Gem programmet som *billede*.

### 2.3 Brug knapperne A og B

En `if` sætning bruges til at lave et valg med, fx skal jeg gå til højre eller venstre. I nedenstående eksempel undersøges om variabelen `x` er større end 4. Hvis den er det, så udfør Ja ellers Nej.

```
if x > 4:
    <Ja>
else:
    <Nej>
```

Efter en `if` sætning skal der være et `:` og linjerne nedenunder skal være rykket ind (TAB).  
Knapper hedder **Buttons** i **Referencen**.

### Opgave

- 1) Lav et nyt projekt
- 2) Brug **Referencen** til at lave et program som skriver **A** når man trykker på knap A og tilsvarende **B** når der trykkes på B.

En variabel kan gemme en værdi som så senere kan bruges i programmet. Værdien bliver typisk ændret mange gang i løbet af programmet og kan fx bruges til at lave en tæller med. En variabel kan have et navn som fx `x`, men kan også have et mere sigende navn som fx `motorhastighed`.

Lav følgende program:

```
from microbit import *

x = 0
while True:
    display.show(x)
    if button_b.was_pressed():
        x += 1
```

Tryk på B knappen og se hvad der sker.

### Opgave

- 1) Brug knap A til at tælle ned med
- 2) Undgå negative tal og tal over 9 (Brug en `if` sætning, det hedder **Logic** i **Referencen**)
- 3) Brug: `display.show(Image.DIAMOND)` til at vise et billede i 1 sekund hvis værdien er mindre end 0 eller større end 9.

Gem filen som *knapper-plus-minus*.

## 2.4 Lav en terning

Brug følgende kode til at lave en terning med:

```
from microbit import *
import random

while True:
    if accelerometer.was_gesture('shake'):
        display.show(random.randint(1, 6))
```

Man 'kaster' terningen ved at ryste den. Prøv om den virker.

### Opgave

- 1) Ændre koden, så man bruger knap A i stedet for at ryste den
- 2) Lav en lille animation, så øjnene skifter hurtigt inden de vises

Gem filen som *terning*.

## 2.5 Reaktionstest

Vi skal nu lave et program som kan måle ens reaktionstid. Vis et billede på lysdioderne og se hvor hurtigt man kan trykke på en knap derefter. Mål hvor lang tid der går mellem billedet vises og der trykkes på knappen. Brug følgende funktion til at aflæse tiden med.

```
import utime

t0 = utime.ticks_ms()
```

t0 er i millisekunder (1/1000 sekunder).

### Opgave

- 1) Vis et billede på lysdioderne
- 2) Lav en forsinkelse inden billedet vises. Her skal du bruge `random.randint` (se Terning øvelsen)
- 3) Gem starttiden t0
- 4) Vent på at knap A bliver trykket på
- 5) Gem sluttiden som t1
- 6) Beregn hvor lang tid der er gået og vis det på lysdioderne
- 7) For at undgå snyd (hvor man bare holder knappen inde), så check inden billedet vises at der **ikke** er trykket på knappen.

Gem program som *knap-reaktion*.

## 2.6 Vaterpas

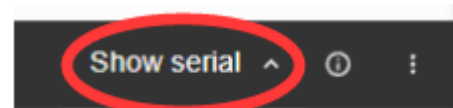
Brug accellerometeret til at lave et vaterpas med. Få en lysdiode til at 'bevæge' sig rundt afhængig af hældningen af micro:bit'en.

Start med følgende kode:

```
from microbit import *

while True:
    x = accelerometer.get_x()
    print(x)
    sleep(200)
```

Tryk på **Show serial** i bunden af skærmen. Tilt micro:bit'en til siderne og se hvordan værdierne ændre sig.



### Opgave

- 1) Slet print sætningen (den var kun til debug)
- 2) Vis en prik på lysdioderne
- 3) Brug `x = int(x/300+2)` til at flytte lysdioden med

Der kommer fejl, når man kommer ud over kanten.

- 4) Tilføj to `if` sætninger så man undgår det. Hint ude til højre.
- 5) Ændre koden så den sletter prikken inden den flyttes

Test nu at prikken kan flyttes fra side til side

```
if x < 0:
    x = 0
```

- 6) Lav nu den tilsvarende kode for y retningen

Test at prikken kan flyttes rundt på hele alle lysdioderne.

Gem program som *vaterpas*.

## 2.7 Skridttæller

Brug **Ideas Step counter** ude til venstre i editoren til at lave en skridt tæller med. Lav også Challenges opgaverne.

## 2.8 Kommunikation

Få to micro:bits til at kommunikere med hinanden. Send en tekst fra den ene micro:bit til den anden. Indholdet kan fx være styres af knapperne på senderen og resultatet vises på modtagerens display. Radiomodul er beskrevet i **Referencen** under **Radio**.

Man kan godt have Editoren kørende to gange på samme tid (i to faner), en til modtager koden og en til sender koden. På den måde kan man arbejde på koden til både sender og modtager på samme tid.

### Opgave

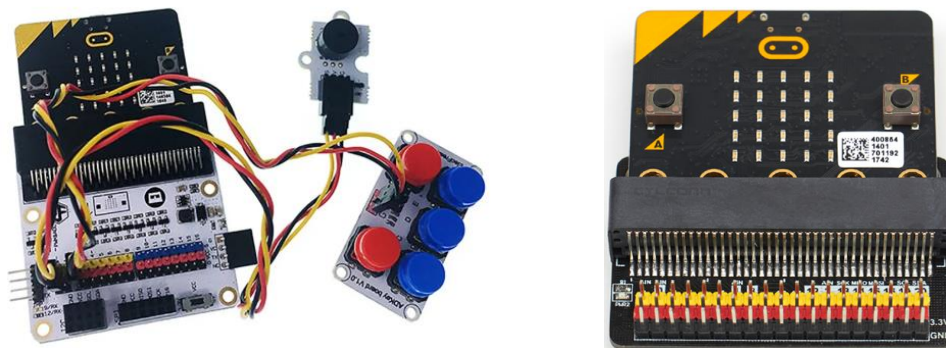
- 1) Brug **Referencen** til at sende en besked fra den ene micro:bit til den anden, når der trykkes på en knap. Hvis I er flere hold der gør det samtidig er det vigtigt at I ikke bruger det samme gruppenummer. Gruppenummeret skal være et tal mellem 0 og 255.
- 2) Prøv at kunne sende forskellige beskeder, ved at bruge begge knapper.

Gem program som *kommunikation*.



## 3 Hardware

I de følgende opgaver skal vi tilslutte nogle hardwareenheder til micro:bit'en. Dette foregår normalt ved at man sætter micro:bit'en i et ekspansionsboard, hvor man så kan tilslutte de hardwareenheder man har brug for. Her er nogle eksempler på ekspansionsboards:



Benene på ekspansionsboardet er både farvede og nummereret.

- **Sort** er 0 (eller ground, GND)
- **Rød** er + (eller plus, VCC)
- **Gul** er signalbenet, som er forbundet til micro:bit'en. De hedder *pins* i koden (og i **Referencen**) og vi skal for det meste bruge `pin0`, `pin1` og `pin2`.

### 3.1 Styr en LED

Vi skal nu tilslutte en lysdiode (LED) og en kontakt som vi skal styres fra micro:bit'en.



#### Opgave

- 1) Tilslut en lysdiode til ben 0. Den skal 'vende' rigtigt, og det ene ben skal sidde på 0 (sort) og det andet på signalbenet (gul).
- 2) Tilføj følgende linje som tænder lysdioden:  
`pin0.write_digital(1)`
- 3) Ændre koden så man kan tænde lysdioden, når der trykkes på knap A
- 4) Ændre koden så man kan slukke lysdioden, når der trykkes på knap B
- 5) Ændre koden så man kan toggle lysdioden, når der trykkes på knap A

Hint: brug en variable og en `if` sætning til at huske om lysdiode skal tændes eller slukkes.

Gem program som *led*.

### 3.2 Styr nogle neopixels

Neopixels er lysdioder som man kan styre farven på. De sidder typisk på en strip eller en ring, hvor der er flere neopixels efter hinanden. De bruger ret meget strøm, så man skal ikke have alt for mange på af gangen.



#### Opgave

- 1) Tilføj en neostrip til ben 0. Den har tre ledninger, og det er vigtigt at de bliver forbundet korrekt.

- 2) Kik i **Referencen** under **NeoPixels** og få første neopixel til at lyse
- 3) Ændre koden så neopixelen lyser i en anden farve. Tallene kan være 0-255.
- 4) Ret antallet af lysdioder i koden, så det matcher antallet af reelle lysdioder
- 5) Lav en loop som tænder alle neopixels
- 6) Brug `random.randint` til at lave tilfældige farver med. Hint: se *terning* opgave.
- 7) Lav et løbelys, hvor en lysdiode 'løber' ned ad strip'en eller rundt i ringen.

Gem program i *neo*.

### 3.3 Alarm hvis den flyttes

Vi skal lave en alarm som tænder, hvis man flytter micro:bit'en.

#### Opgave

- 1) Tilføj en buzzer til ben 0
- 2) Få buzzeren til at hyle når micro:bit'en flyttes. Hint: man kan se om den flyttes ved at aflæse accellerometeret. Kik på koden for *vaterpas*.
- 3) Får også alle lysdioderne til at blinke når alarmen aktiveres

Gem program i *alarm*.

### 3.4 Tilslut andre enheder

#### Opgave


- 1) Find en eller flere sjove enheder og tilslut dem til micro:bit'en
- 2) Lav noget sjovt med dem

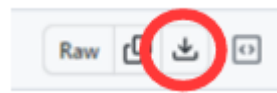
## 4 Bit:Bot bil

### 4.1 Bit:Bot Python modul

For at gøre det lettere at programmere Bit:Bot bilen har jeg lavet et modul ligesom fx til display'et. Da modulet ikke er indbygget i Python, skal det manuelt tilføjes første gang det skal bruges. Man skal først hente filen: bitbot.py.

- 1) Tryk på denne link: <https://github.com/mhfalken/microbit/blob/main/bitbot.py>

- 2) Tryk på  ude til højre (Download raw file)



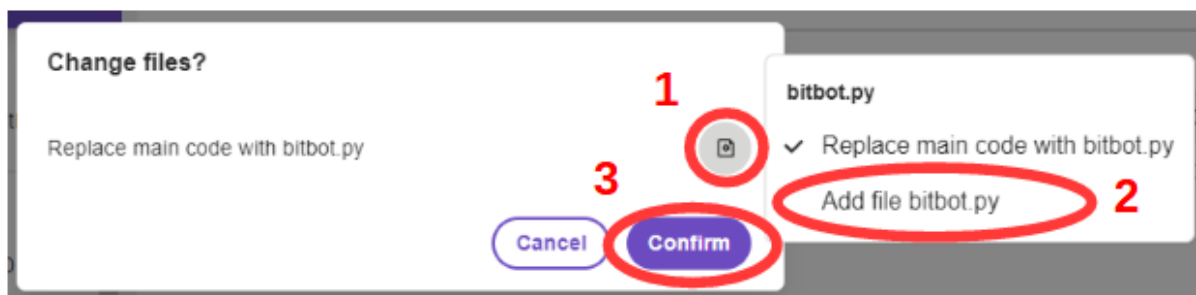
- 3) Filen ligger nu i **Downloads** eller **Overførsler** folderen.


Man tilføjer filen til sit projekt på følgende måde:

- 4) Tryk på **Open** i bunden ude til højre og vælg *bitbot.py* filen som du lige har downloaded (se billede).



Derefter kommer følgende dialog frem:



- 5) Tryk først på  (1), derefter vælg **Add File bitbot.py** (2) og tryk til sidst på **Confirm** (3).  
Nu er filen tilføjet til dit projekt. (Modulet håndterer både den klassiske bil og XL bilen automatisk.)

For at bruge modulet skal det importeres lige som alle andre moduler.

```
import bitbot  
  
bb = bitbot.bitbot()
```

Selve Bit:Bot modulet er beskrevet sidst i dokumentet, hvis man vil se alle funktionerne, men dem vi skal bruge bliver beskrevet efterhånden som de bliver brugt.

Hver gang man starter på et nyt projekt som skal bruge biblioteket, skal det tilføjes som vist overfor (kun step 4+5).

### 4.2 NeoPixels

På bilen sidder der en række NeoPixels (lysdioder) langs begge sider (6 på hver side). De har alle et nummer som står ved siden af NeoPixel'en.

Lav et nyt program og indtast følgende for at tænde NeoPixel 0. (Husk at tilføje bitbot.py filen som beskrevet ovenfor)

```
from microbit import *
import bitbot

bb = bitbot.bitbot()
np = bb.NeoPixel()
np[0] = (25, 25, 25)
np.show()
```

De lyser meget kraftigt, så man kan med fordel bruge værdierne 0-25. (255 er max)

Farven angives i RGB (rød, grøn, blå). Så `np[0] = (25, 0, 0)` vil give en rød farve.

- 1) Ændre koden til at tænde NeoPixel 7
- 2) Ændre farven, prøv flere farver
- 3) Tænd alle NeoPixels en af gangen. Brug: `for i in range(12):`
- 4) Lav et sejt lysshow i farver – brug eventuelt tilfældige tal som er beskrevet nedenfor.

#### 4.2.1 Tilfældige tal (random)

Hvis man gerne vil have en tilfældig farve, kan man bruge en funktion som laver et tilfældig tal.

`randint(0, 25)` giver et tilfældig tal mellem 0 og 25.

Det kan fx bruges sådan her:

```
from microbit import *
import bitbot
import random

bb = bitbot.bitbot()
np = bb.NeoPixel()
np[0] = (random.randint(1, 50), random.randint(1, 50), random.randint(1, 50))
np.show()
```

Gem program som *bil-neo-show*.

### 4.3 Firkant

Vi skal nu bruge funktionen `Drive(left, right)` til at styre motorerne med, og den tager værdier mellem -100 og 100, hvor -100 er fuld fart bak, 0 er stop og 100 er fuld fart frem. Alle heltal kan bruges, så 50 er halv fart frem.

`Drive(50, 50)` får bil til at køre fremad med halv fart. Det vil være klogt når I skal lave jeres programmer at I sørger for at bilen stopper når programmet er færdigt. Fx slut jeres programmer med:

```
bb.Drive(0, 0)
```

- 1) Lav et program som får bilen til at køre fremad i 2 sekunder
- 2) Lav et program som får bilen til at køre til venstre i 2 sekunder
- 3) Lav et program som får bilen til at køre i en firkant. Det vil kræve nogle test at få en 'pæn' firkant.
- 4) Sæt en tusch på bilen og tegn firkanten på et stykke papir. Undgå at tegne på gulvet!

Gem program som *bil-firkant*.

### 4.4 Følg linje

Linjesensorerne aflæses med følgende funktion:

```
if bb.ReadLine(bitbot.LEFT): # Venstre sensor ser sort stribe
    ...
```

- 1) XL bilen har to lysdioder som viser om sensoren ser striben. Se at de virker ved at bevæge bilen hen over en sort streg (med hånden).
- 2) Får motorerne til at køre langsomt frem og se at lysdioderne kan se strengen.
- 3) Få den motor hvis lysdiode lyser til at køre lidt langsommere for at komme tilbage på linjen. Husk man kan godt bruge negative værdier.
- 4) Trim hastighederne så bilen følger linjen – det kræver nogle forsøg 😊 Det er lettere hvis bilen kører langsomt.

Hint: Det er vigtigt at bilen kan dreje meget skarpt, dvs. at det ene hjul kører baglæns mens det andet kører forlæns, men samtidig er det vigtigt at bilen sammenlagt bevæger sig fremad, da den ellers kan 'hænge' fast i et skarpt sving. Dette opnås ved at den 'bakker' mindre end den kører fremad ( $-10 + 14 > 0$ ).

- 5) Tegn jeres egen bane.

Gem program som *bil-linje*.

## 4.5 Sonar

Denne opgave kræver en sonar som er ekstra udstyr til bilen. Sonaren sættes forrest på bilen og kan måle afstanden til en forhindring i cm.

```
afstand = bb.Sonar()
```

- 1) Udvid 'følg linje' så bilen stopper når den ser en forhindring.
- 2) Tilføj NeoPixels som blinker alt efter hvad bilen laver.
- 3) Brug buzzer. `bb.Buzzer(value)`, value = 0 eller 1.
- 4) Brug også displayet til noget (lysdioderne)

Sonaren opfanger også lidt støj, så der kommer en nogle 'fejl' målinger, hvilket godt kan give lidt problemer. Tip: Hvis man kalder Sonar for ofte, så risikere man at modtage tidligere ekkoer, da det kan bevæge sig ganske langt. Så det er en god ide kun at kalde det nogle gange i sekundet.

Gem program som *bil-linje-sonar-bip*.

## 5 Appendiks – ekstra opgaver

Her er nogle ekstra opgaver som er lidt sværere, da der ikke er så meget hjælp til dem.

### 5.1 Bit:Bot bil

#### 5.1.1 Kørt efter lyset

Få bilen til at køre efter lyset, ved at lyse på lyssensorerne på oversiden af bilen foran. Brug funktionen: `ReadLight(sensor)` til at aflæse lysstyrken for hver sensor.

#### 5.1.2 Find vej i en labyrinth

Få bilen til at finde vej i en labyrinth, ved at køre til højre hver gang den ser en forhindring.

#### 5.1.3 Fjernstyret bil

Lav en fjernstyret bil vha. af to micro:bits, en som fjernbetjening og en i bilen. Brug ideer fra `vaterpas.py` og `kommunikation.py` til at styre motorerne med.

Her er flere udfordringer:

- Det kræver to forskellige programmer
- Der skal laves en 'protokol' til at sende informationer med. Overvej hvad sender modulet skal lave og hvad modtager modulet skal lave og dermed hvilke informationer som det vil være smart at overføre. Det vil være en god ide at lave en funktion i sender modulet.
- Der skal bruges mange teknikker

Hints til protokollen:

Data sendes som en streng, og det vil derfor være en fordel med et fast format:

"M: <data><data>"

```
radio.send("D:%4i:%4i" %(left, right))
```

En streng er en liste, så når data modtages i den anden ende, kan det dekodes med:

```
if besked[0] == 'D':  
    left = int(besked[2:6])  
    right = int(besked[7:])
```

Hvis man har brug for flere forskellige beskeder, så ændres start bogstavet.

Hvis der ikke er nogen besked, så har man ikke nogen streng, så programmet vil dø! Dette løses med:

```
if type(besked) != str:  
    continue
```

Her checkes om beskeden er en streng inden man eventuelt dekoder den.

Gem filerne som *rc-bil-sender* og *rc-bil-modtager*.

#### 5.1.3.1 Månefartøj

Indbyg forsinkelse i sendermodulet, så det ser ud som om bilen er på månen og styringen foregår fra jorden. For at lave forsinkelsen skal man lave en kø og til det bruger man en liste, hvor man putter kommandoerne ind i den ene ende og tager dem ud lidt senere af den anden ende og sender dem til bilen. Prøv nu at styre bilen igennem en bane.

### 5.2 Ideer til videre forløb

#### 5.2.1 Projekter

Andre ideer til projekter.

- Mål fugtighed på pin0 (brug fingrene)

- LED UV meter, graf!
- Storage
- Eksterne porte
  - Servo

## 6 Appendiks – info

Dette afsnit er en blanding af ekstra information, løsninger og ideer til fremtidige programmer.

### 6.1 bitbot.py

Her er en beskrivelse af bitbot.py modulet og hvilke funktioner som det stiller til rådighed.

Modulet har generelt ingen kommentarer for at spare på FLASH forbruget. Modul konstanterne ligger uden for 'klassen' for at gøre RAM forbruget mere fleksibelt.

```
MODEL_XL
MODEL_CLASSIC
LEFT
RIGHT
```

`Drive(left, right)`

Motor styring. Left, right er hastigheden for hver motor [-100; 100], hvor -100 er fuld bak, 0 er stop og 100 fuld fart frem. Alle heltal kan bruges.

`ReadLine(sensor)`

Læser linjesensorerne som sidder på undersiden af bilen. Sensor er LEFT eller RIGHT. Funktionen returnerer True hvis mørk linje under sensor.

`ReadLight(sensor)`

Læser lyssensorerne, som sidder på oversiden af bilen helt fremme. Sensor er LEFT eller RIGHT. Værdien er 0-100, hvor 100 er fuldt lys.

`Buzzer(value)`

Tænd og sluk buzzer. Value er 1 eller 0.

`NeoPixel()`

Returnere et NeoPixel objekt som passer til bilen.

`ModelGet()`

Returnerer biltypen, MODEL\_CLASSIC eller MODEL\_XL.

`Sonar(max=100)`

Sonar er ekstra udstyr, som kan måle afstanden foran bilen. Den returnerer afstanden i cm. Max er indført for at begrænse hvor lang tid en måling kan tage. Max = 100 betyder at den stopper med at måle hvis afstanden er over 100 cm.

### 6.2 Løsninger

Her er løsningsforslag til nogle af opgaverne.



### led-fill.py

```
from microbit import *

for x in range(5):
    for y in range(5):
        display.set_pixel(x, y, x+y)

# x retning
# y retning (plus nedad!)
```

### knapper-plus-minus.py

```
from microbit import *

x = 0
while True:
    display.show(x)
    if button_b.was_pressed():
        x += 1
        # Knap B tæller op
    if button_a.was_pressed():
        x -= 1
        # Knap A tæller ned
    if x < 0:
        x = 0
        # Undgå negative tal
        display.show(Image.DIAMOND)
        sleep(1000)
    if x > 9:
        x = 9
        # Undgå tal over 9
        display.show(Image.CHESSBOARD)
        sleep(1000)
```

### terning.py

```

from microbit import *
import random

while True:
    if button_a.was_pressed():
        for i in range(7):
            display.show(random.randint(1, 6))
            sleep(50)

```

#### knap-reaktion.py

```

from microbit import *
import random
import utime

x = 0
while True:
    sleep(random.randint(1000, 2000))
    if button_a.is_pressed():
        display.show(Image.ANGRY)
        sleep(1000)
        display.clear()
    else:
        display.show(Image.DIAMOND)
        t0 = utime.ticks_ms()
        while True:
            if button_a.is_pressed():
                t1 = utime.ticks_ms()
                break

        display.scroll(t1-t0)

```

#### vaterpas.py

```

from microbit import *

while True:
    x = accelerometer.get_x()
    x = int(x/300+2)
    if x < 0:
        x = 0
    if x > 4:
        x = 4
    y = accelerometer.get_y()
    y = int(y/300+2)
    if y < 0:
        y = 0
    if y > 4:
        y = 4
    display.set_pixel(x,y,9)
    sleep(200)
    display.set_pixel(x,y,0)

```

### bil-neo-show.py

```
from microbit import *
import bitbot
import neopixel
from random import *

bb = bitbot.bitbot()
np = bb.NeoPixel()
while True:
    i = randint(0, 11) # Tilfældig neopixel
    np[i] = (randint(0, 50), randint(0, 50), randint(0, 50)) # Tilfældig farve
    np.show() # Vis neopixels
```

### bil-firkant.py

```
from microbit import *
import bitbot

bb = bitbot.bitbot()
sleep(1000)
for i in range(4):
    bb.Drive(50, 50)
    sleep(1000)
    bb.Drive(-35, 35)
    sleep(850)
bb.Drive(0, 0)
```

### bil-linje.py

```
from microbit import *
import bitbot

bb = bitbot.bitbot()

while True:
    if bb.ReadLine(bitbot.LEFT): # Left ser striben
        bb.Drive(-10, 14)
    elif bb.ReadLine(bitbot.RIGHT): # Right ser striben
        bb.Drive(14, -10)
    else:
        bb.Drive(18, 18)
```

### bil-linje-sonar-bip.py

```
from microbit import *
import bitbot

bb = bitbot.bitbot()
np = bb.NeoPixel()

if bb.GetModel() == bitbot.MODEL_XL:
    display.show("X")
else:
    display.show("C")

bb.Buzzer(1)
sleep(500)
bb.Buzzer(0)
i = 0

while True:
    np[5] = np[11] = (0, 0, 0)
    if bb.ReadLine(bitbot.LEFT):      # Left ser striben
        bb.Drive(-10, 14)
        np[11] = (0, 10, 0)
    elif bb.ReadLine(bitbot.RIGHT):   # Right ser striben
        bb.Drive(14, -10)
        np[5] = (0, 10, 0)
    else:
        bb.Drive(18, 18)
    np[0] = np[6] = (i % 10, i % 10, 0)
    np.show()
    if bb.GetModel() == bitbot.MODEL_XL:
        while bb.Sonar() < 10:
            bb.Drive(0, 0)
            np[5] = np[11] = (15, 0, 0)
            np.show()
    i += 1
```

### kommunikation.py

```
from microbit import *
import radio

radio.config(group=2)
radio.on()

while True:
    if button_a.was_pressed():
        radio.send('A')
        display.show("A")
    elif button_b.was_pressed():
        radio.send('B')
        display.show("B")
    incoming = radio.receive()
    if incoming == 'A':
        display.show("A")
    elif incoming == 'B':
        display.show("B")
    sleep(500)
    display.show("-")
```

**rc-bil-sender.py**

```

from microbit import *
import radio

def Drive(speed, direction):
    if speed < 0:
        direction = -direction
    left = speed + direction
    right = speed - direction
    if left > 100:
        left = 100
    elif left < -100:
        left = -100
    if right > 100:
        right = 100
    elif right < -100:
        right = -100
    radio.send("D:%4i:%4i" %(left, right))

radio.config(group=2)
radio.on()

l = [0, 0]
while True:
    if button_a.was_pressed():
        radio.send("A")

    l[0] = accelerometer.get_x()
    l[1] = accelerometer.get_y()
    speed = -int(l[1]/10)
    if abs(speed) < 5:
        speed = 0

    direction = int(l[0]/20)
    if abs(direction) < 5:
        direction = 0

    for i in range(2):
        l[i] = int(l[i]/300 + 2)
        if l[i] < 0:
            l[i] = 0
        elif l[i] > 4:
            l[i] = 4

    Drive(speed, direction)
    display.set_pixel(l[0], l[1], 9)
    sleep(100)
    display.set_pixel(l[0], l[1], 0)

```

**rc-bil-modtager:**

```

from microbit import *
import radio
import bitbot

bb = bitbot.bitbot()
np = bb.NeoPixel()
display.show('R')

def Leds(offset, v):
    for i in range(5):
        np[i+offset] = (0, 0, 0)
    if v > 0:
        c = (0, 50, 0)
    else:
        c = (50, 0, 0)
    if abs(v) > 80:
        np[4+offset] = c
    if abs(v) > 60:
        np[3+offset] = c
    if abs(v) > 40:
        np[2+offset] = c
    if abs(v) > 20:
        np[1+offset] = c
    if abs(v) > 0:
        np[0+offset] = c

bb.Drive(0, 0)
radio.config(group=2)
radio.on()
while True:
    incoming = radio.receive()
    if type(incoming) != str:
        continue
    if incoming[0] == 'D':
        left = int(incoming[2:6])
        right = int(incoming[7:])

        bb.Drive(left, right)
        Leds(0, left)
        Leds(6, right)
        np.show()

    if incoming == "A":
        bb.Buzzer(1)
        sleep(200)
        bb.Buzzer(0)

```

### 6.2.1 FLASH og RAM forbrug

Micro:bit'en (V1) har kun 256 kbyte flash og 16 kbytes RAM. For de fleste små programmer, har man ikke noget problem, men hvis man bliver grebet af at kode, så kan man godt løbe tør for plads.

Fejlkoden er dog meget den samme for både flash og RAM – MemoryError. Her er lidt tips til at spare på resurserne:

Flash: kommentarer fylder i flash, så start med at slette dem. Det samme gælder tomme linjer. Brug TAB i stedet for spaces.

Hvis man ikke skal bruge bitbot.py modulet, kan det med fordel slettes for at spare flash.

RAM: `range(100)`, laver en liste i RAM med 100 elementer, så en løkke som  
`for i in range(100):` bruger meget RAM. Brug i stedet:

```
i=0
while i < 100:
    i +=1
```

Husk generelt at variable i Python fylder meget mere end deres reelle plads behøver.



## 7 Python reference

Her er en kort oversigt over nogle af de mest brugte Python konstruktioner.

### 7.1 Generelle

#### 7.1.1 Kommentarer

Kommentarer bruges til at skrive tekster, så det er lettere at forstå koden. De kan også bruges til at slå noget kode fra pga. test.

```
# Dette er en kommentar

""" Start af kommentar

slut på kommentar"""
```

#### 7.1.2 Import

Import bruges til at importere kode fra andre moduler. Al micro:bit kode skal starte med at importere micro:bit modulet, men vi bruger også andre moduler. `from xxx import yyy`, betyder at man kun importere dele af modulet.

```
from microbit import *
import neopixel
from utime import ticks_us, sleep_us
```

#### 7.1.3 Print

Print bruges til at skrive ud på seriellporten med (REPL). For micro:bit bruges dette primært til at debugge sin kode med.

```
print(x)          # Print værdien af x
```

#### 7.1.4 Pass

`pass` gør ingenting, og bruges derfor sjældent, men kan være rar at have alligevel.

```
for i in range(2):
    pass          # Gør ingenting
```

## 7.2 Datatyper

Datatyper bruges til at definere ens variable med. Python har flere forskellige datatyper, som bruges til forskellige formål. Det er typisk afgørende at vælge den rigtige datatype for at få ens kode til at blive så simpel som mulig. Det vil være god praksis at begynde variable navne med små bogstaver!

#### 7.2.1 Simple typer

`<navn> = <værdi>`, opret en simpel variabel.

```
x = 1              # Sæt x lig med 1
a = 'hej'          # Lav en streng
x += 1             # Læg 1 til x
```

## 7.2.2 Tupler

Tupler er en ordnet mængde af elementer. Antallet af elementer kan ikke ændres og det samme gælder for værdierne. De defineres med bøjet parenteser `()`. Indeks starter fra 0.

`<navn> = (<elementer>)`

```
t = (2, 4)      # Lav tuple med 2 elementer
print(t[0])     # Print første element
```

## 7.2.3 Lister

Dette er en list af elementer, hvor antallet af elementer kan ændres over tid dvs. man kan indsætte og fjerne elementer. Værdierne kan også ændres. De defineres med skarpe parenteser `[]`. Indeks starter fra 0.

`<navn> = [<elementer>]`

Operatorer (der findes mange flere):

`<list>.append(<element>)`, tilføj et element til enden af listen

`<list>.remove(<element>)`, fjern et element fra listen

`len(<list>)`, find antallet af elementer

```
l = []          # Lav en tom liste
l = [1, 2, 6]   # Lav en liste med 3 elementer
l = range(3)    # Lav liste med 3 elementer 0, 1, 2
l.append(23)    # Tilføj 23 til enden af listen
l.remove(6)     # Fjern 6 fra listen
print(len(l))  # Print antallet af elementer
print(l[0])    # Print første element
l[1:2]         # Udtag liste med element 1
```

## 7.3 Kontrol

Disse konstruktioner bruges til at styre programflowet med.

### 7.3.1 If/else

Hvis 'betingelse' er sand gør A ellers B. `else` grenen er frivillig.

if `<betingelse>`:

`<indrykket kode>`

else:

`<indrykket kode>`

```
if x > 3:
    print("hej")
else:
    print("dov")
```

### 7.3.2 Try/except

Dette er en speciel konstruktion som bruges til at fange hvis noget går galt, dvs man kan undgå at ens kode 'dør'.

try:

`<kode som kan dø>`

except:

`<hvis koden døde, så gør det her>`

```
try:
    i2c.read(0x1C, 1)[0]
except:
    print("fejl")
```

## 7.4 Loops

Loops bruges til at gentage den samme kode flere gange.

### 7.4.1 While

Så længe 'betingelse' er sand, så forsæt løkken.

```
while <betingelse>:
    <indrykket kode>
```

```
while True:
    print("hej")
```

# Loop for evigt

### 7.4.2 For

Dette er en loop for et bestemt antal iterationer.

```
for <variable> in <list>:
    <indrykket kode>
```

```
for i in range(5):
    print("hej")
```

# Loop fra 0 til 4

### 7.4.3 Break, continue

Hvis man gerne vil forlade en loop 'før' tid, så kan man bruge følgende:

**break** exit loop nu

```
for i in range(5):
    if i > 3:
        break
    print("hej")
```

# Loop fra 0 til 4  
# Forlad loop'en før tid

**continue** fortsæt med 'næste' loop, dvs spring resten af loop'en over.

```
for i in range(5):
    print("dav")
    if i > 3:
        continue
    print("hej")
```

# Loop fra 0 til 4  
# Undlad print("hej") hvis i > 3

## 7.5 Funktioner

Funktioner bruges til at opdele sin kode i genbrugelige blokke, som så kan kaldes fra flere forskellige steder. Det vil være god praksis at begynde funktionsnavne med Store bogstaver. Funktioner kan returnere alt fra ingenting til en liste af argumenter.

```
def <navn>(parameter, ..., parameter=<værdi>):
    <indrykket kode>
    return <værdi>
```

```
def PlusTal(a, b, c=0):      # Definer funktion, c er valgfri
    return a+b+c

x = PlusTal(1, 2, 3)        # Angiv alle parametre
y = PlusTal(3,6)            # Angiv kun 2 værdier, c bliver så 0

def PlusFlereTal(a, b, c):
    Return a+b, a+c        # Her returneres to resultater

x, y = PlusFlereTal(1, 4, 10)
```