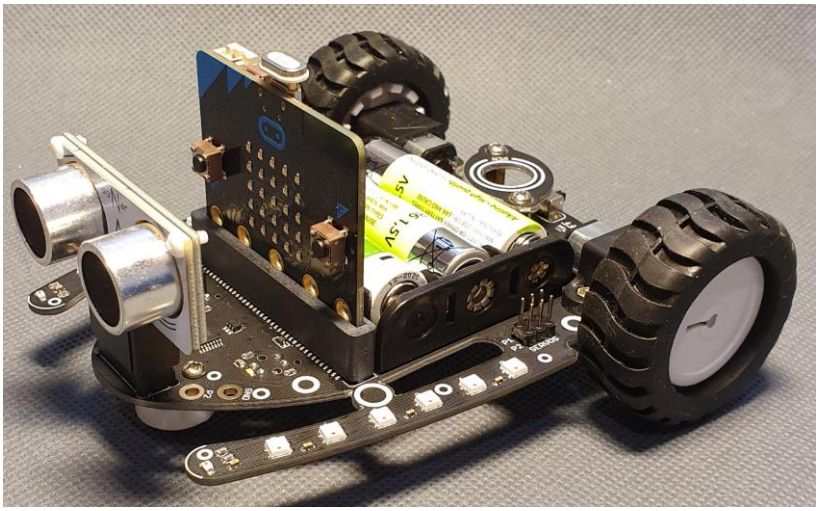
Python med micro:bit og bit:Bot

Af: Michael Hansen, Coding Pirates Furesø, 2020, version 1.32
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. Selve dokumentet er baseret på 'learning by doing', dvs der er ikke meget Python undervisning i selve dokumentet. Jeg har lavet et præsentationssæt som gennemgå de vigtigste ting i Python og som også indeholder lidt generelt viden omkring embedded systemer.
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 års alderen som ønsker at eksperimentere med en bil og/eller ønsker at lære Python.

1 Introduktion 3
1.1 Installering 3
1.2 Hello world 3
1.3 Basic debug 4
2 micro:bit 5
2.1 Vis et billede 5
2.2 Test knapper, op/ned tælling 5
2.3 Kompas 6
2.4 LED 6
3 Bit:Bot bil 7
3.1 Bit:Bot Python modul 7
3.2 NeoPixels 7
3.3 Firkant 8
3.4 Følg linje 8
3.5 Sonar 9
4 Appendiks – ekstra opgaver 10
4.1 micro:bit alene 10
4.2 micro:bit og Bit:Bot bil 11
5 Appendiks – info 12
5.1 bitbot.py 12
5.2 Løsninger 12
5.3 Debug tips 18
5.4 Ideer til videre forløb 19
6 Python reference 20
6.1 Generelle 20
6.2 Datatyper 20
6.3 Kontrol 21
6.4 Loops 22
6.5 Funktioner 22

1 Introduktion

1.1 Installering

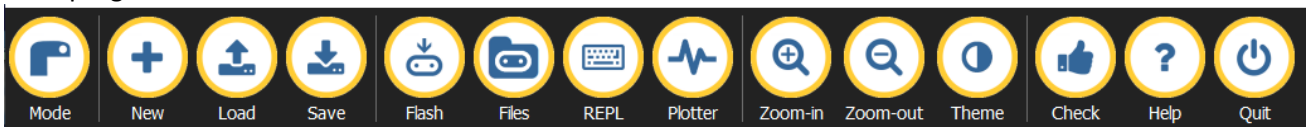
Man kan programmere micro:bit'en på to måder:

- web editor
- programmet MU

MU editoren skal installeres på PCen og har nogle flere muligheder som gør den både lettere at bruge og det er også lettere at debugge sit program. Jeg anbefaler derfor klart at man installerer MU i stedet for at bruge webversionen.

1.1.1 MU installering

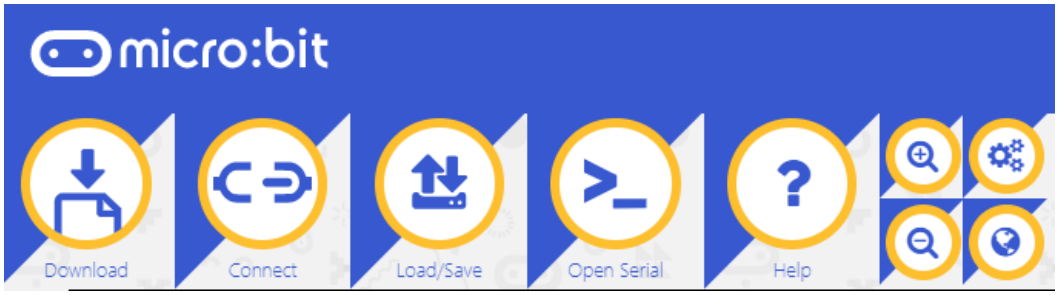
Download MU fra denne hjemmeside: https://codewith.mu/en/download
Installer programmet – det tager nogle minutter.
Start programmet MU.



Start med at trykke på Mode knappen og vælg BBC micro:bit.
Første gang man Flash'er et program, lægger MU automatisk Python ned på micro:bit'en også. Der kan godt komme en fejl første gang, så prøv bare igen.

1.1.2 Web installering

Før man kan bruge webversionen, skal man manuelt opdatere micro:bit'en med Python. Følg vejledning på denne hjemmeside: https://microbit.org/guide/firmware/ (den er på engelsk, men det er ganske let!)
Når man har opdateret micro:bit'en, så åbnes editoren: https://python.microbit.org/v/beta



Tryk først på Connect og vælg BBC micro:bit.
I denne editor hedder REPL knappen "Open Serial" (bruges senere).

1.2 Hello world

Indtast følgende, brug "tab-completions". Husk Python kode er case-sensitiv (store og små bogstaver er forskellige!)

```
from microbit import *

while True:
    display.scroll('Hello')
    sleep(2000)
```

Tryk Flash/Download (ikon i toppen af editoren):
Gennemgang af kode.

- import denne linje skal altid være der i alle programmer man laver

- while True loop for evigt
- Indrykning Rigtig indrykning er vigtig i Python, brug TAB
- display.scroll Viser tekst på LEDs
- sleep – ms Venter i x millisekunder (1/1000 sekund)

Gem fil (Save) som **helloworld.py.**
Det er generelt vigtigt at gemme ens programmer, da man kan genbruge mange dele af koden senere hen.

Opgave

1) Ret tekst 'Hello' til noget andet

1.3 Basic debug

Ved tryk på 'Check' knap (kun MU editor), checker editoren om koden ser rigtig ud. Hvis noget er galt bliver det tydeligt markeret. Alle fejl skal fjernes før programmet virker.
Kommentarer bruges til lettere at forstå programmet, men kan også bruges til at fjerne en linje midlertidig.

```
# Dette er en kommentar
```

REPL/Open Serial – live kommandoer
Tryk på REPL og prøv nogle kommandoer:

```
1+1
display.scroll("Dav")
```

Dette kan bruge til hurtigt at teste nogle kommandoer med. Det kan også bruges til debug hvilket vises senere.
Tryk på REPL for at komme tilbage til editoren (toggler mode).

2 micro:bit

2.1 Vis et billede

Man kan vise et billede på LED med følgende kode. Lav nyt program (New) og indtast følgende program.

```
from microbit import *

display.show(Image.DIAMOND)
```

Man kan vælge mellem flere forskellige billeder – editoren kan hjælpe her.

1) Prøv nogle af de andre billeder

Man kan også lave sine egne billeder med følgende linjer.

```
img = Image('99999:07770:00500:03330:11111')
display.show(img)
```

Tallene 0-9 angiver lysstyrken, de første 5 tal er linje 1, næste 5 linje 2 osv.

1) Lav jeres eget billede

Gem som **led-billede.py**.

2.2 Test knapper, op/ned tælling

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`.
En `if` sætning bruges til at lave et valg med, fx skal jeg gå til højre eller venstre. I nedenstående eksempel undersøges om variablen `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).
Lav nyt program og indtast 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
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 program som **knapper-plus-minus.py.**

2.3 Kompas

Her vil vi bruge kompasset til at viser hvordan man kan skrive noget ud på terminalen – primært til debug brug. Man skriver ud på terminalen med `print("Hej")`.
Nyt program og indtast følgende:

```
from microbit import *

display.show(Image.DUCK)
while True:
    print(compass.heading())
```

Flash koden. Kompasset skal først kalibreres. Dette gøres ved at bevæge det rundt i alle retninger til alle LEDs lyser.
Tryk på REPL og derefter Ctrl-D!

Opgave

1) Hvad vises på terminalen?
2) Ændre koden så den ikke skriver så hurtigt

Man kommer ud af REPL modet ved at trykke på REPL knappen. Dette kan bruges til debug, hvis man gerne vil vide hvad en værdi er eller se hvorfor et program stopper.
Gem program som **kompas.py**.

2.3.1 Find Nord

Fjern print sætningen og lav et kompas som kan finde nord. Nord er når `compass.heading()` er større end 350 og mindre end 10. Nedenstående linje kan med fordel bruges:

```
if h > 350 or h < 10:
```

1) Vis N på displayet når nord er fundet
2) Hvordan kan man gøre kompasset mere præcist?

Gem program som **kompas-nord.py**.

2.4 LED

Indtil nu har vi set en loop som kører for evigt – `while True`. Vi skal nu lave en loop som kører et vist antal gange. For `x in range(5):` giver værdierne x= 0, 1, 2, 3, 4. Læg mærke til at start værdien er 0!
Funktionen `display.set_pixel(x, y, value)` tænder én LED på displayet. `x` og `y` er koordinater med (0, 0) i venstre øverste hjørne og `value` er lysstyrken 0-9.
Nyt program og indtast følgende:

```
from microbit import *

for x in range(5):
    display.set_pixel(x, 0, 5)
```

1) Ændre 5 til 9 – hvad sker der?
2) Tænd linje 2 i stedet for linje 0
3) Lav endnu en loop (brug variable `y`) som tænder alle LED rækker
4) Ændre `9` til `x+y`

Gem program som **led-fill.py**.

3 Bit:Bot bil

Da man har brug for at tage USB kablet af og på hele tiden anbefaler jeg at man anskaffer en ledning med et magnetisk micro-usb-stik.

3.1 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 MicroPython, skal det manuelt installeres første gang det skal bruges. Man skal først hente filen: bitbot.py. (https://github.com/mhfalken/microbit). Få en voksen til at hjælpe.
MU: Filen bitbot.py skal ligge i <user>/MU_code/. Brug derefter Files knappen til at kopiere filen over på micro:bit'en (drag and drop).
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 beskrivelse sidst i dokumentet, hvis man vil se alle funktionerne, men dem vi skal bruge bliver beskrevet efterhånden som de bliver brugt.

3.2 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.

Nyt program og indtast følgende for at tænde NeoPixel 0.

```
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 `for i in range(12):`
4) Lav et sejt lysshow i farver – brug eventuelt tilfældige tal som er beskrevet nedenfor.

3.2.1 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.

```
from random import *

x = randint(0, 25)
```

Det kan fx bruges sådan her:

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

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

Gem program som **bil-neo-show.py**.

3.3 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.py**.

3.4 Følg linje

Linjesensorerne aflæses med følgende funktion:

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

1) XL bilen har to LEDs 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 LED'erne kan se stregen.
3) Få den motor hvis LED 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.
5) Tegn jeres egen bane.

Gem program som **bil-linje.py**.

3.5 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 (LEDs)

Sonaren opfanger desværre en del støj, så der kommer en del 'fejl' målinger, hvilken godt kan give lidt problemer.

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

4 Appendiks – ekstra opgaver

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

4.1 micro:bit alene

4.1.1 Reaktionstest

Lav et program, hvor man kan måle ens reaktionstid. Vis et billede på LEDs 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 måle tiden med.

```
import utime

t = utime.ticks_ms()
```

t er tiden i millisekunder (1/1000 sekunder).
Gem program som **knap-reaktion.py**.

4.1.2 Terningspil

Lav et program hvor man kan 'kaste' med en terning. Man kan vælge at vise et tal, eller man kan viser 'øjnene'. Man kan vælge at trykke på en tast eller man kan 'ryste' micro:bit'en. Man kan også lave en terning med flere øjne end 6. Her er lidt inspiration…

```
import random

random.randint(1, 6)
accelerometer.was_gesture("shake")
```

4.1.3 Kuglespil

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

```
accelerometer.get_x()
accelerometer.get_y()
```

Start med at printe værdierne ud for at få et overblik over hvordan de virker.
Gem program som **kuglespil.py**.

4.1.4 Kommunikation

Få to micro:bits til at kommunikere med hinanden. Send nogle bytes fra den ene micro:bit til den anden. Indholdet kan fx være styres af knapperne på senderen og resultatet vises på modtagerens display. Radiomodulet bruges på følgende måde:

```
import radio

radio.config(group=2)    # Set gruppe til 2 (vælg selv et tal)
radio.on()

radio.send('hej')        # Beskeden er 'hej'
incoming = radio.receive()
```

Vælg et gruppenummerer, som de andre ikke bruger. Det skal være et tal mellem 0 og 255.
Prøv at kunne sende flere forskellige beskeder, ved at bruge begge knapper.
Gem program som **kommunikation.py**.

4.2 micro:bit og Bit:Bot bil

4.2.1 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.

4.2.2 Find vej i en labyrint

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

4.2.3 Fjernstyres bil

Lav en fjernstyret bil vha. af to micro:bits, en som fjernbetjening og en i bilen. Brug ideer fra kuglespil.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.py** og **rc-bil-modtager.py**.

5 Appendiks – info

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

5.1 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.

5.2 Løsninger

Her er løsningsforslag til de fleste af opgaverne.

knapper-plus-minus.py

```
from microbit import *

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

kompas.py

```
from microbit import *

while True:
    h = compass.heading()
    if h > 355 or h < 5:
        display.show("N")
        sleep(500)
    else:
        display.clear()
```

led-fill.py

```
from microbit import *

for x in range(5):                      # x retning
    for y in range(5):                  # y retning (plus nedad!)
        display.set_pixel(x, y, x+y)
```

kompas-nord.py

```
from microbit import *

while True:
    h = compass.heading()
    if h > 355 or h < 5:
        display.show("N")
        sleep(500)
    else:
        display.clear()
```

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

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).

```
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
```

knap-reaktion.py

```
from microbit import *
import random
import utime

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

    t = utime.ticks_ms() - t
    display.scroll(t)
```

kuglespil.py

```
from microbit import *

l = [0, 0]
while True:
    l[0] = accelerometer.get_x()
    l[1] = accelerometer.get_y()
    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
    display.set_pixel(l[0], l[1], 9)
    sleep(100)
    display.set_pixel(l[0], l[1], 5)
    if button_a.was_pressed():
        display.clear()
```

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)
```

5.3 Debug tips

Her er lidt tip til at debugge sin kode med.
Hvis man bruger MU editor så er det en god ide at bruge Check knappen til at se om koden ser lovlig ud. Det er dog ikke nok til at sikre at koden er korrekt, men et godt sted at starte.
Hvis micro:bit'en skriver en fejlbesked ud på skærmen, så kan man med fordel trykke på REPL knappen og derefter på Ctrl-D, så vil fejlbeskeden komme ud på skærmen incl linje nummer og forklaring. Man kan også printe variable ud for bedre at forstå hvad programmet gør.

5.3.1 FLASH og RAM forbrug

Micro:bit'en 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.

5.4 Ideer til videre forløb

5.4.1 Projekter

Andre ideer til projekter.

- Mål magnet felt
- Mål fugtighed på pin0 (brug fingrene)
- LED UV meter, graf!
- Storage
- Eksterne porte
    - Servo
- Plotter (print tuple, MU editor)

6 Python reference

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

6.1 Generelle

6.1.1 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"""
```

6.1.2 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
```

6.1.3 Print

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

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

6.1.4 Pass

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

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

6.2 Datatyper

Datatyper bruges til at definer 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 begynder variable navne med små bogstaver!

6.2.1 Simple typer

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

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

6.2.2 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
```

6.2.3 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>)`, tilfæj 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 elemet 1
```

6.3 Kontrol

Disse konstruktioner bruges til at styre program flowet med.

6.3.1 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(”dav”)
```

6.3.2 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 døede, så gør det her>

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

6.4 Loops

Loops bruges til at gentage den samme kode flere gange.

6.4.1 While

Så længe ’betingelse’ er sand, så forsæt løkken.
while <betingelse>:
  <indrykket kode>

```
while True:                 # Loop for evigt
    print(”hej”)
```

6.4.2 For

Dette er en loop for et bestemt antal iterationer.
for <variable> in <list>:
  <indrykket kode>

```
for i in range(5):          # Loop fra 0 til 4
    print(”hej”)
```

6.4.3 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):          # Loop fra 0 til 4
    if i > 3:
        break               # Forlad loop’en før tid
    print(”hej”)
```

`continue` fortsæt med ’næste’ loop, dvs spring resten af loop’en over.

```
for i in range(5):          # Loop fra 0 til 4
    print(”dav”)
    if i > 3:
        continue            # Undlad print(“hej”) hvis i > 3
    print(”hej”)
```

6.5 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)
```