Kobling B

Programmering og skaperverksted for 8. - 10. klasse

Kapittel 12

Innhold

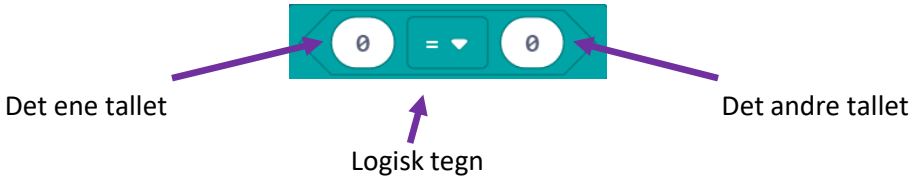
Kapittel 12 – Introduksjon til Pythonprogrammering

Introduksjon til Python-programmering

Vilkår (conditions)

Vilkår i Python er det samme som vilkår i Scratch eller MakeCode. Forskjellen mellom dem er at i stedet for å ha ferdige sekskanta klosser med vilkår, må vi skrive vilkårene selv i Python. Et vilkår er et utsagn som kan være sant eller usant, og den typen vilkår vi kan bruke i programmering vil alltid sammenligne noe. Ofte sammenligner vi tallverdier, og ser om de er like store eller det første er mindre enn det andre, eller omvendt. Det går også an å sammenligne tekster for å se om de er like eller ikke. Men for tekster kan man ikke finne ut om den ene teksten er større enn en annen. Hvordan skulle vi målt det?

Et vilkår i python må bestå av tre deler, akkurat som i Scratch og MakeCode. To tall som skal sammenlignes på en eller annen måte, og et tegn som sier hvordan de skal sammenlignes. Dette tegnet kalles ofte for et logisk tegn eller en logisk operator. Dersom sammenligningen stemmer, vil vilkåret gi oss verdien «sann» tilbake. Dersom sammenligningen ikke stemmer, vil vi i stedet få verdien «usann» tilbake. Sann og usann kalles boolske verdier. Det går an å ha boolske variable, det vil si at variabelen kan lagre verdien sann eller usann.



Under kommer en oversikt over hvordan man kan skrive forskjellige vilkår. Se om du klarer å finne ut om eksemplene returnerer verdien sann eller usann.

Vilkår		
Tegn	**Forklaring**	**Eksempler**
>	Større enn	4 > 5
<	Mindre enn	2 < 7
>=	Større enn eller lik	-2 >= 0
<=	Mindre enn eller lik	7 <= 12
==	Lik	9 == 19
!=	Ikke lik	3 != 6
or	Ett av vilkårene må være sanne	3 == 4 or 3 > 1
and	Alle vilkårene må være sanne	3 == 4 and 3 > 1

Regning

En del ganger har vi behov for å utføre regnestykker i Python. Dette er spesielt nyttig der vi har noen variabler vi, for eksempel, skal multiplisere eller legge sammen. Men det går fint an å utføre regnestykker direkte med tall. Under er en oversikt over forskjellige regneoperatorer for Python. Se om du klarer å se hva verdien av regnestykke-eksemplene blir.

Regneoperatorer		
Tegn	**Forklaring**	**Eksempler**
+	Addisjon	(-4) + 5
-	Subtraksjon	(-1) - (-4)
*	Multiplikasjon	3 * 0
/	Divisjon	12 / (-3)
**	Opphøyd i (eksponent)	2**3

Variabler i Python

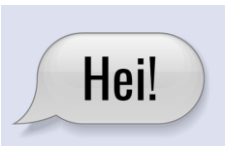
Ulike typer variabler er viktige innen programmering. Ett av poengene med å lage programmer, er at vi skal kunne bruke det samme programmet på å løse flere lignende oppgaver. Da er det mye greiere å lagre alle tall vi skal bruke i variabler, slik at vi bare trenger å bytte ut tallene ett sted, for at programmet likevel skal gi oss riktig svar.

En variabel kan inneholde én verdi, og hver gang vi legger en verdi i en variabel, erstatter vi den verdien som eventuelt ligger i variabelen fra før. Når vi lager en variabel i Python, trenger vi ikke skrive hvilken type det er, det gjenkjenner Python ut fra den første verdien vi legger inn i variabelen. Det finnes fire typer variabler:

Heltall
- integer

Desimaltall
- float

Tekst
- string



Boolsk verdi
- boolean

Heltall kalles integer i Python. En integer er et positivt eller negativt tall som ikke har noen desimaler. 0 er også en integer.

Desimaltall kalles float i Python. En float er et tall som inneholder desimaler. Det kan være både negativt eller positivt. Tallet 2,0 er en float.

Tekst kalles string i Python. (På norsk blir også ordet «streng» brukt.) En tekstverdi er ett eller flere ord, det kan også være hele setninger. For at python skal forstå at det er en tekstverdi, må vi bruke apostrofer (‘) rundt teksten. Dersom vi vil legge ordet hei inn i variabelen a, må vi skrive kommandoen slik: `a = 'hei'`

Boolsk verdi kalles boolean value i Python. Denne verdien kan være enten sann eller usann. Den blir brukt i forbindelse med vilkår, siden et vilkår kan være enten sant eller usant.

Av og til trenger vi å gjøre om en type variabel til en annen type variabel. Slik som når det er en funksjon i Python som bare tar inn heltall. Dersom vi da har et desimaltall, må vi gjøre det om til et heltall, og da må vi avrunde desimaltallet vårt. Da bruker vi kommandoen round(). Det går også an å bruke kommandoen int(), men den avrunder ikke desimaltallet, den bare kutter ut alt bak kommaet. Det betyr i praksis at desimaltallet alltid blir rundet ned til nærmeste heltall.

For å kunne sende tall mellom to eller flere micro:biter, må verdiene være gjort om til tekstverdier. For å gjøre om en tallverdi til en tekstverdi kan vi bruke kommandoen str().

Oppgave
Hva vil vises på skjermen for disse programmene? Kan du lage noen programmer selv? Få en i klassen til å gjette hva som vil skje i programmene dine, eller kanskje læreren?

```
a = 10
print(a)
a = a + 1
print(a)
```

```
a = 1.7
a = round(a)
print (a)
```

```
a = '1.7'
a = float(a)
print(a+a)
```

```
a = 1.7
a = str(a)
b = float(a)
print(a+'b')
```

```
a = 1.7
a = int(a)
print(a)
```

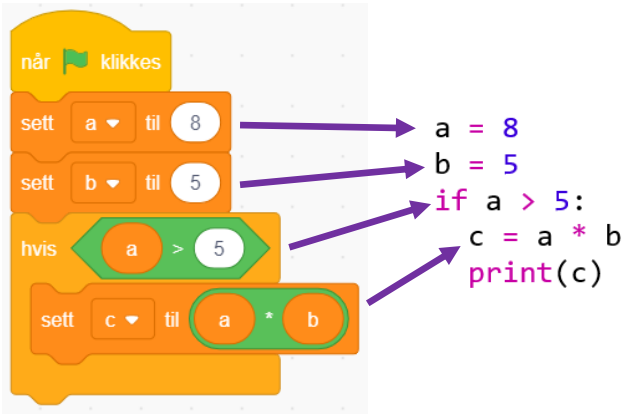
```
a = '1.7'
a = float(a)
print(a)
```

```
a = '1.7'
print(a+a)
```

```
a = 1.7
#a = str(a)
b = float(a)
print(a+b)
```

Hvis-setninger (if-setninger)

Hvis-setninger tilsvarer hvis-klosser i Scratch og MakeCode. De må inneholde et vilkår, og dersom vilkåret er sant, vil det som «tilhører» hvis-setningen bli utført. I de klossbaserte programmeringsspråkene putter vi inn det som skal skje ved hjelp av klossene som passer slik som puslespillbrikker. I Pyton må vi gjøre det samme, bare at vi må skrive alt selv. Siden vi nå ikke har klosser for å vise hva som står «inni» hvis-setningen, bruker vi innrykk i stedet. Alt som står etter en hvis-setning, med et innrykk foran, vil «stå inni» hvis-setningen, og blir bare utført om vilkåret i hvis-setningen er sant.



Disse to programmene gjør det samme, og hver blokk i Scratch tilsvarer én linje i Python, med to unntak.

I Python trenger vi ikke å trykke på et flagg, vi trykker på en «play-knapp» som får koden til å begynne å kjøre. Dermed trenger vi ikke noe som tilsvarer den første blokka i Scratch-koden.

I Python viser ikke verdien på variablene, med mindre vi skriver dem ut til skjermen ved å bruke print-kommandoen. I Scratch vises verdien til alle variablene hele tiden, så der trenger vi ikke å få vist fram c på noen spesiell måte.

Ofte har vi behov for å ha flere alternativer som kan være sanne. Det finnes to forskjellige varianter, og de kalles gjerne «ellers» eller «ellers hvis».

Ellers – det som skal skje, dersom vilkåret i hvis-setningen er usant.

```
else:
```

Ellers hvis – det som skal skje dersom vilkåret i hvis-setningen er usant og følgende vilkår er sant.

```
elif:
```

Snakk om

Hva måtte du endre i Python-programmet for at det bare skulle regne ut c når verdien i b var større enn verdien i a?

Hvilken forskjell hadde det blitt dersom vi flyttet linja med print(c) helt til venstre i Pythonkoden, slik at den ikke sto med innrykk?

Oppgave

Hva vil vises på skjermen for disse programmene? Kan du lage noen programmer selv? Få en i klassen til å gjette hva som vil skje i programmene dine, eller kanskje læreren?

```
a = 8
b = 5
if a > 5:
  c = a * b
  print(c)
else:
  print(a)
  print(b)
```

```
a = 8
b = 5
if a > b:
  c = a * b
  print(c)
elif a < b:
  print(a)
  print(b)
```

```
a = 5
b = 8
if a > b:
  c = a * b
  print(c)
elif a==b:
  print(a)
else:
  print(a)
  print(b)
```

```
a = 5
b = a
if a > b:
  c = a * b
  print(c)
elif a==b:
  print(a)
else:
  print(a)
  print(b)
```

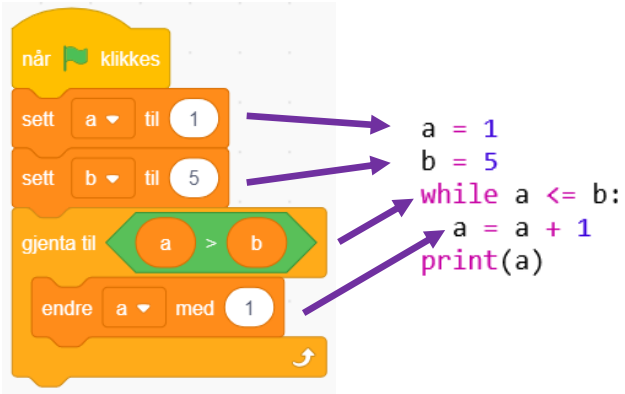
```
a = 5
b = 8
if a > b:
  c = a * b
  print(c)
elif a==b:
  print(a)
elif a < b:
  print(a)
  print(b)
```

Hvordan skrive hvis-setninger – en oppskrift

```
if betingelse:
    kode som utføres når betingelsen er sann
elif betingelse:
    kode som utføres når betingelsen er sann og betingelsen over er usann
else:
    kode som utføres når ingen av de andre betingelsene er sanne
```

While-løkker i Python

While-løkker (vilkårløkker) tilsvarer «gjenta til-løkkene» i Scratch. De må ha med et vilkår som forteller om det som står inni løkka skal gjøres. For Scratch er det slik at det som står inni løkka utføres helt til vilkåret blir sant (altså at løkka går mens vilkåret er usant). I Python er det motsatt, at det som står inni løkka blir utført helt til vilkåret ikke lenger er sant. Det gjør at selve vilkåret må skrives på forskjellige måter, dersom man skal oppnå det samme med Scratch og Python.



Disse to programmene gjør det samme, og hver blokk i Scratch tilsvarer én linje i Python, men med to unntak.

I Python trenger vi ikke å trykke på et flagg, vi trykker på en «play-knapp» som får koden til å begynne å kjøre. Dermed trenger vi ikke noe som tilsvarer den første klossen i Scratch-koden.

I Python vises ikke verdien på variablene, med mindre vi skriver dem ut til skjermen ved å bruke print-kommandoen. I Scratch vises verdien til alle variablene hele tiden, så der trenger vi ikke å få vist fram a på noen spesiell måte, men vi rekker ikke å se hvilke verdier a har underveis, siden programmet går så raskt.

Snakk om

Hva vil vise på skjermen for Python-programmet?

Vilkåret i de to programmene er ikke like. Hva ville Pythonprogrammet vist, dersom man brukte vilkåret fra Scratch-programmet direkte?

Hvilken verdi ville a hatt, dersom vi brukte vilkåret fra Pythonprogrammet direkte i Scratch-programmet?

Hvordan skrive while-løkker – en oppskrift

```
while vilkår:
  kode som utføres mens vilkåret er sant
```

Pass på å sette alt som skal være inni while-løkka med innrykk. Dersom det ikke står med innrykk, vil det ikke skje før løkka er ferdig, og programmet fortsetter videre.

Oppgave

1. Hva vil vises på skjermen for disse programmene?

Skriv ned hva du tror vil vises på skjermen, og skriv inn programmene med Pythonkode etterpå. Kjør koden, og sjekk om du får det svaret du trodde du skulle få.

2. Dersom du ikke fikk svaret du trodde, hva var grunnen til det, tror du? Diskuter gjerne med andre.
3. Kan du lage noen programmer selv? Få en i klassen til å gjette hva som vil skje i programmene dine, eller kanskje læreren?

```
a = 1
b = 5
while a < b:
  a = a + 1
  print(a)
```

```
a = 1
b = 5
while a > b:
  a = a + 1
  print(a)
```

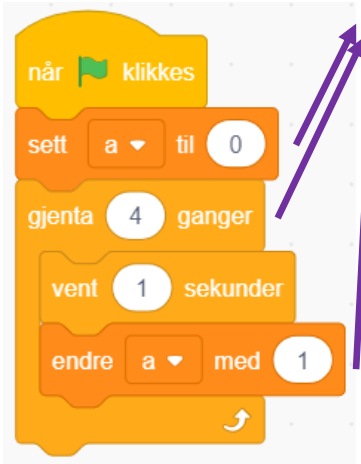
```
a = 10
b = 0
while a > b:
  a = a - 1
  print(a)
```

```
a = 10
b = 0
while a > b:
  a = a - 1
print(a)
```

```
a = 1
b = 5
while a < b:
  a = a + 1
  print(a)
  print (b)
```

For-løkker i Python

For-løkker (telleløkker) tilsvarer «gjenta et visst antall ganger-løkkene» i Scratch. Disse løkkene gjentar det som står inni dem så mange ganger som vi ønsker. I Scratch kan vi bare skrive direkte hvor mange ganger vi vil gjenta løkka, men i Python er det litt annerledes. Der blir det opprettet en variabel som heter i, vi kan gjerne kalle det en tellevariabel. Den begynner på den tallverdien som står først i parentesen til for-løkka, i vårt eksempel er det 0. Det andre tallet i parentesen viser hvilket tall tellervariabelen i skal telle seg opp til (men ikke inkludert). I vårt tilfelle skal i telle seg opp til verdien 5. Det siste tallet inni parentesen sier hvor store steg vi skal øke verdien til tellevariabelen i med. I vårt eksempel skal vi telle med 1, altså sier vi at tellervariabelen begynner på 0, vi teller opp til 5 ved å øke med 1 hver gang. Da vil i være 0, 1, 2, 3, 4 før løkka blir avslutta.




```
for i in range (0, 5, 1):
  print(i)
```

Snakk om

Hva ville skjedd om vi brukte variabel a i stedet for tallet 4 i løkka i Scratch?

Går både Scratch-programmet og Pythonprogrammet gjennom løkka like mange ganger?

Hvordan skrive for-løkker – en oppskrift

```
for i in range (fra og med-verdi, til-verdi, steglengde):
  kode som utføres mens i teller gjennom verdiene den skal ha
```

Disse to programmene gjør det samme, men her tilsvarer ikke hver kloss i Scratch én linje i Python. Her er tre av blokkene i Scratch innbakt i for-løkka i python. I tillegg er det to klosser i Scratch som ikke tilsvarer noen linjer i Python.

I Python trenger vi ikke å trykke på et flagg, vi trykker på en «play-knapp» som får koden til å begynne å kjøre. Dermed trenger vi ikke noe som tilsvarer den første klossen i Scratch-koden.

I Python viser ikke verdien på variablene med mindre vi skriver dem ut til skjermen ved å bruke print-kommandoen. I Scratch vises verdien til alle variablene hele tiden, så der trenger vi ikke å få vist fram a på noen spesiell måte, men vi rekker ikke å se hvilke verdier den har underveis, siden det går så raskt. For å kunne se hvilke verdier variabelen i Scratch har, så har vi denne gang lagt inn en kloss som sier «vent 1 sekunder», og den gjør at programmet tar en pause i ett sekund. Da rekker vi å se verdien på variabelen før programmet går videre igjen.

Oppgave

1. Hva vil vises på skjermen for disse fem programmene?

Skriv ned hva du tror vil vises på skjermen, og skriv inn programmene med Pythonkode etterpå.
Kjør koden, og sjekk om du får det svaret du trodde du skulle få.

2. Dersom du ikke fikk svaret du trodde, hva var grunnen til det, tror du? Diskuter gjerne med andre.
3. Kan du lage noen programmer selv? Få en i klassen til å gjette hva som vil skje i programmene dine, eller kanskje læreren?

```
for i in range (0, 5, 2):
  print(i)
```

```
for i in range (0, 5, 0):
  print(i)
```

```
for i in range (1, 10, 2):
  print(i+i)
```

```
for i in range (1, 10, 2):
  print(i-i)
```

```
for i in range (1, 10, 1):
  print(i**2)
```

Lister i Python

Ofte trenger vi å bruke mange variabler, og da kan det være greit å lage en liste eller en tabell (array i Python). Dersom én variabel tilsvarer én skuff som har et navn og en verdi, så er en liste som en kommode med flere skuffer. Hver variabel er ett element i lista, og hvert element har et navn og en verdi. Navnet på elementet er navnet på hele lista sammen med plasseringen på lista, altså hvilken rad det tilhører.

Vi trenger dermed ikke å lage et eget navn til hvert element. Det er også enkelt å bruke en løkke for å gå gjennom alle elementene for å bruke dem i, for eksempel et regnestykke.

En liste har et navn og elementer med hvert sitt nummer. Det som er litt spesielt med lister innen programmering, er at det første elementet er element nummer 0. Det er fort gjort å glemme når man programmerer, og er alltid lurt å sjekke som en del av feilsøkingen. En liste har i utgangspunktet bare elementer med én datatype. Det går ikke an å bruke noen heltall og noen tekstverdier i samme liste. En liste er i utgangspunktet en lang rekke med elementer, og er dermed endimensjonal.

Navnet på lista må være ett sammenhengende ord for at Python skal forstå det. Innen programmering er det vanlig å bruke _ som mellomrom for navn på variabler, lister og tabeller. Bokstavene æ, ø og å bør vi unngå, siden noen Python-editorer ikke takler navn som bruker andre enn engelske tegn.

Ønskeliste

0. Hund
1. Mobil
2. Sko
3. Bukse
4. Hagenisse

Et element har både et navn og en verdi.

Hva er navnet og hva er verdien til dette elementet?

Oversikt over noen nyttige kommandoer i Python

```
liste = []
```
Lager en tom liste.

```
liste = [1, 3, 5, 7]
```
Lager en liste med verdiene 1, 3, 5, 7.

```
liste[3] = 7
```
Legger inn verdien 7 i element tre, i en liste med minst fire elementer.

```
del liste[3]
```
Sletter element tre, i en liste med minst fire elementer.

```
liste.append(4)
```
Legger til verdien fire som et nytt element i en liste som heter liste. Det har ingenting å si hvor mange elementer som er i lista fra før, for verdien blir alltid lagt i et nytt element etter det siste elementet i den opprinnelige lista.

```
print(liste[2])
```
Henter verdien til element to i lista, og skriver det på skjermen.

```
len(liste)
```
Finner hvor mange elementer det er i en liste.

Diskusjonsoppgaver

1. Hvorfor er det liten vits i å bruke en løkke for å legge sammen mange variabler?
2. Hvorfor er det enkelt å bruke en løkke for å gjøre det samme dersom du har alle verdiene i en liste?

Tabeller i Python

En tabell (array) kan inneholde flere typer data, og den kan være todimensjonal. Det vil si at den har mer enn én kolonne med data. Da trenger vi to tall for å forklare hvilket element i tabellen vi prater om. Det blir på samme måte som et punkt i et koordinatsystem, der vi må bruke både x- og y- koordinaten for å plassere et punkt. Det første elementet begynner på 0 for tabeller også, både for radene og for kolonnene. Tabeller finnes i numpy-biblioteket, og må importeres: `from numpy import*`

Rene tekstfiler (.txt-filer) fra statistisk sentralbyrå gir todimensjonale (2D) tabeller i Python, og er dermed viktige når vi skal analysere data, og lage matematiske modeller. Det er ganske enkelt å gjøre om en 2D- array til en 1D-array som er det fit_function() tar inn av data (se kapittel 13).

Tabellen under har navnet budsjett. Akkurat som for lister, begynner vi å telle på null. Dette gjelder for både radene (vannrett) og kolonnene (loddrett). Rad null i budsjett-tabellen inneholder tekstverdiene: Måned, Inntekt, Klær og snacks, Mobilutgifter, Diverse utgifter, SUM. Kolonne null inneholder Måned, Januar, Februar, Mars, April, Mai, Juni.

Måned	Inntekt	Klær og snacks	Mobilutgifter	Diverse utgifter	SUM
Januar	1500	1000	300	100	100
Februar	1500	1000	300	100	100
Mars	3000	1900	300	100	700
April	1500	1000	300	100	100
Mai	1500	1000	300	100	100
Juni	2500	1000	300	300	900

Dette er element [3,5] og har verdien 700.

Diskusjonsoppgaver

1. Hvilken verdi har element [0,0]?
2. Hvilket element har verdien 3000?

Oversikt over noen nyttige kommandoer

```
budsjett[7][2] = 2000
```

Endrer verdien fra 1500 til 2000 i elementet i rad syv og kolonne to i budsjett-tabellen på denne sida.

```
print(budsjett[3][1])
```

Skriver ut verdien til elementet i rad tre og kolonne en, altså 1500 fra budsjettet på denne sida.

```
tabell = zeros([2,3])
```

Lager en tabell med to rader og tre kolonner, der alle verdiene er null.

```
tabell = array([(1, 2, 3),(4, 5, 6)])
```

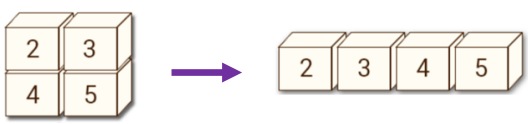
Lager en tabell med to rader og tre kolonner. Tallene innenfor parentesene angir verdiene til elementene i hver rad.

```
tabell = linspace(0, 5, 11)
```

Lager en 1D-tabell med 11 verdier jevnt fordelt fra og med null til og med fem.

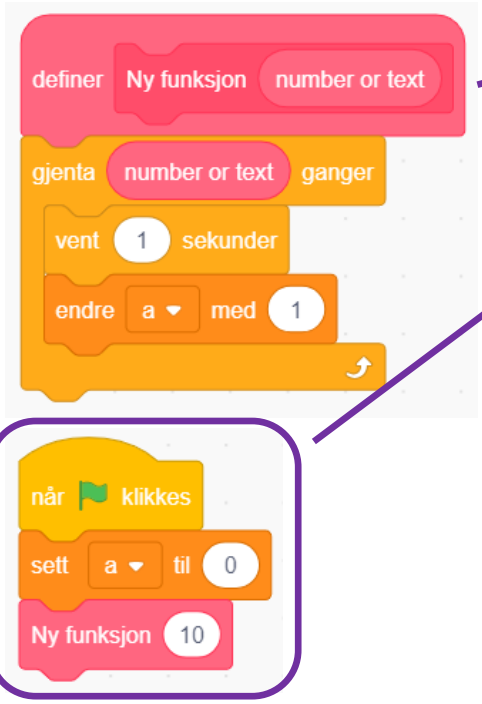
```
a = tabell.flatten()
```

Gjør om en 2D-tabell til en 1D-tabell. Alle radene i 2D-tabellen blir lagt etter hverandre til en lang rad.



Funksjoner i Python

Funksjoner i Python tilsvarer «lag en ny kloss» i Scratch, og er egentlig bare småprogrammer. Faktisk trenger en funksjon ikke være veldig liten heller, men det som skiller den fra et vanlig program, er at den må bli definert før den kan brukes. Når funksjonen er definert, så kan vi bruke den så mange ganger vi ønsker uten å skrive all koden på ny, bare ved å skrive én kodelinje.




```
def Funksjon(inndata):
  for i in range(0, inndata+1, 1):
    print(i)

Funksjon(10)
```

Disse to programmene gjør det samme, men her tilsvarer ikke hver kloss i Scratch én linje i Python. Funksjonen som blir definert, inneholder en for-løkke som ikke er bygd opp på helt lik måte i Scratch og Python (se for-løkke siden).

I Python trenger vi ikke å trykke på et flagg, men vi må legge inn en linje for å vise variabelen på skjermen. For å kunne se alle verdiene på skjermen, har vi lagt inn en pause på ett sekund i Scratch-programmet.

Både i Scratch og Python kan funksjonene motta inn-verdier. Det vil si at når vi bruker funksjonen, så må vi oppgi én eller flere verdier som blir plassert i variabler. Disse verdiene blir da brukt til noe i funksjonen, gjerne en utregning. I eksempelet heter variabelen til inn-verdien i Pythonprogrammet inndata.

En viktig forskjell på funksjoner i Scratch og Python, er at i Python kan funksjonene gi oss en ut-verdi. Det vil si at vi kan for eksempel bruke en funksjon til å regne ut et regnestykke for oss, og gi oss et tall tilbake. Dette tallet kan vi for eksempel putte inn i nye regnestykker, i hvis-setninger eller i løkker. I Scratch går ikke dette an.

Snakk om

Hvorfor må vi ha med at vi skal sette variabelen a til 0 i Scratch-programmet?

Går både Scratch-programmet og Pythonprogrammet gjennom løkka like mange ganger?

Oppgave

1. Hva vil vises på skjermen for disse tre programmene?

Skriv ned hva du tror vil vises på skjermen, og skriv inn programmene med Pythonkode etterpå.
Kjør koden, og sjekk om du får det svaret du trodde du skulle få.

2. Dersom du ikke fikk svaret du trodde, hva var grunnen til det, tror du? Diskuter gjerne med andre.
3. Kan du lage noen programmer selv? Få en i klassen til å gjette hva som vil skje i programmene dine, eller kanskje læreren?

```
def Funksjon(inndata):
  return inndata*2

Funksjon(1)
```

```
def Funksjon(inndata):
  return inndata*2

print(Funksjon(8.5))
```

```
def Funksjon(inndata):
  return inndata**2

for i in range(1, 11, 1):
  print(Funksjon(i))
```

Hvordan definere og bruke funksjoner – en oppskrift

```
def navn(variabler):
  kode som utføres av funksjonen
  return returverdien til funksjonen

navn(variabler)
```

Tegning i Python - turtle

Python har et bibliotek som heter turtle, som kan brukes når man ønsker å bruke Python til å tegne figurer eller mønstre. For å importere dette biblioteket, må man skrive: `from turtle import *` øverst i programmet.

Eksempler på noen grunnleggende funksjoner fra turtle-biblioteket

```
t = Turtle()
```

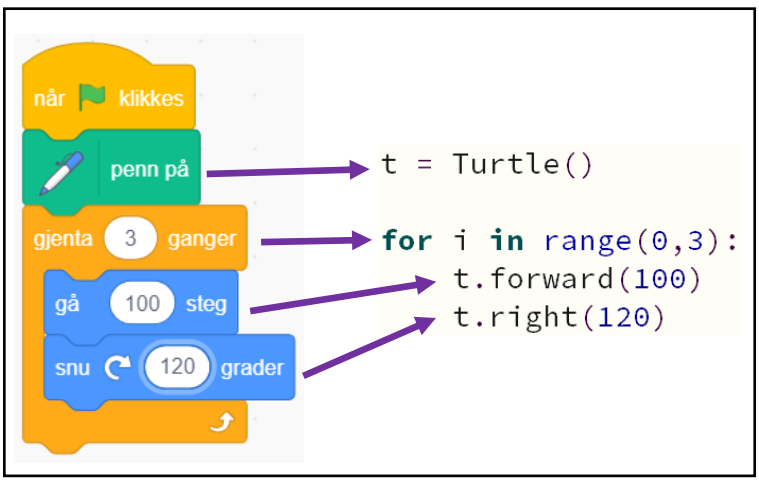
Denne funksjonen lager et «turtle-objekt» som man trenger når man bruker de andre funksjonene som lager tegningen. Vi kan tenke på det som en blyant eller penn.

```
t.forward(100)
t.backward(100)
```

Disse funksjonene flytter turtle frem eller tilbake 100 steg.

```
t.left(60)
t.right(60)
```

Disse funksjonene snur turtle 120 grader til venstre eller til høyre (det er det samme som at den ytre vinkelen er 60 grader.




```
t = Turtle()
for i in range(0,3):
    t.forward(100)
    t.right(120)
```

```
t.circle(50)
```

Denne funksjonen tegner en sirkel med radius 50 steg.

```
t.goto(100,150)
```

Denne funksjonen får turtle til å bevege seg til punktet med x-koordinat 100 og y-koordinat 150.

Oppgave

1. Hva vil vises på skjermen for disse programmene?

Skriv ned hva du tror vil vises på skjermen, og skriv inn programmene med Pythonkode etterpå. Kjør koden, og sjekk om du får det svaret du trodde du skulle få.

2. Dersom du ikke fikk svaret du trodde, hva var grunnen til det, tror du? Diskuter gjerne med andre.

3. Kan du lage noen programmer selv? Få en i klassen til å gjette hva som blir tegnet med programmene dine, eller kanskje læreren?

```
t = Turtle()
for i in range(0,4):
    t.forward(100)
    t.right(90)
```

```
t = Turtle()
for i in range(0,36):
    t.forward(10)
    t.left(10)
```

```
t = Turtle()
for i in range(0,36):
    t.right(10)
    for i in range(0,36):
        t.forward(10)
        t.left(10)
```

Oversikt over noen kommandoer for micro:bit i Python Mu

Når vi skal bruke Python for å programmere en micro:bit, finnes det en del ferdige kommandoer vi kan bruke. Her viser vi en oversikt over noen av de mest nyttige. Andre eksempler kan du finne på denne nettsiden: https://microbit-micropython.readthedocs.io/en/v1.0.1/tutorials/introduction.html

Vi må alltid begynne med å importere det biblioteket som inneholder micro:bit-kommandoene. Det gjør man ved å skrive øverst: `from microbit import *`

Inndata

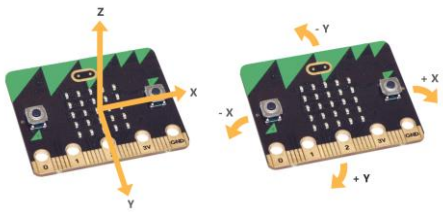
Knapper

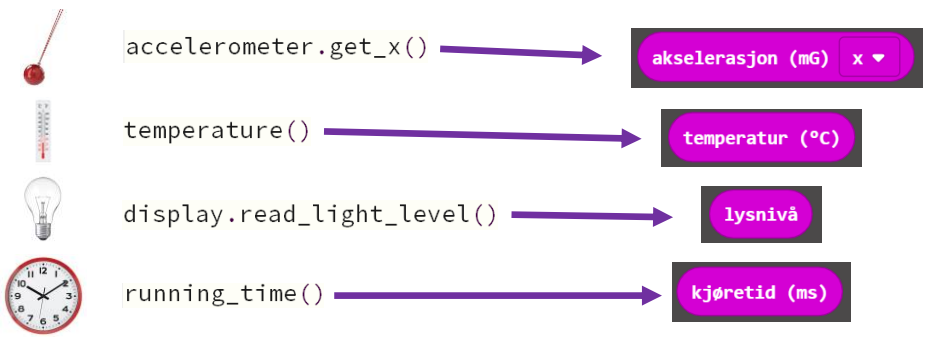
Siden micro:biten har to knapper, A og B, så kan den delen av koden der det står button_a i eksemplene byttes ut med button_b i stedet.

Denne kommandoen teller antall ganger en knapp har vært trykket siden programmet startet.

Innebygde sensorer

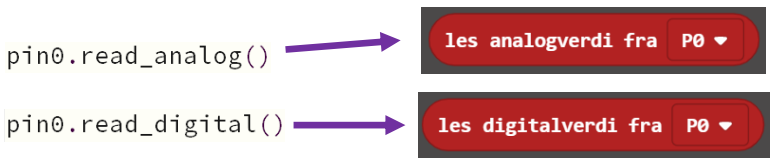
Vi kan bruke micro:bitens innebygde sensorer med Python-programmering også. For akselerasjonen, kan denne måles i alle de tre romlige retningene vi har. Tenk på det som et koordinatsystem med 3 akser. De vanlige, x-aksen og y-aksen, men også en z-akse som står loddrett på begge de to andre aksene.





Påkoblede sensorer – se også siden om sonar

En digital verdi kan bare være enten 1 eller 0, der 1 betyr på og 0 betyr av. Den digitale verdien 1 er egentlig en høy spenning, mens den digitale verdien 0 er en veldig lav spenning. Digitale verdier bruker vi sjelden for sensorer, bare dersom vi skal sjekke om noe er avskrudd eller påskrudd. Sensorer vi kobler på micro:biten gir som regel verdier mellom 0 og 1023, og da må vi lese av en analog verdi i stedet. En analog verdi kan være mye annet enn bare 0 eller 1, og passer dermed best for å lese av hvilken verdi sensoren gir.



Utdata

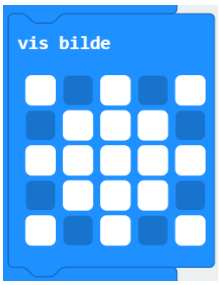
Skrive til skjermen

Det går an å skru på de enkelte lysdiodene på skjermen til micro:biten, med verdier mellom 0 og 9 for hvor sterkt de skal lyse.

```
display.show()
```


```
display.show(temperature())
```


```
display.show(Image("90909:09990:99999:09990:90909"))
```



Påkoblede lysdioder

Når du kobler på en lysdiode til micro:biten, kan du programmere den til å skru seg av eller på. Da må man bruke kommandoen som sender ut digital data, altså verdien 1 eller 0, der 1 tilsvarer på og 0 tilsvarer av.

Hvilken utgang lysdioden er tilkoblet (0, 1 eller 2).

```
pin0.write_digital(1)
pin0.write_digital(0)
```

På (1) eller av (0).

Sende data over radio-forbindelsen

På samme måte som ved å bruke klosser, kan vi programmere micro:biter til å sende data til hverandre ved å bruke Python.

```
import radio
radio.on()
```

Import radio-kommandoen importerer biblioteket som gjør at vi kan bruke radio-forbindelsen mellom micro:bitene. I tillegg må vi bruke radio.on() som skrur på radioforbindelsen.

Til slutt må vi definere hvilken kanal micro:bitene skal kommunisere på. Da må vi passe på å velge samme kanal for alle micro:bitene som skal «snakke sammen».


```
radio.config(channel=1)
```


```
x = str(temperature())
radio.send_bytes(x)
```

Når man programmerer i Python, må man gjøre om tall til tekst (strenger) før det går an å sende dem over radio-forbindelsen. Det gjøres med str()-kommandoen.


```
x = radio.receive_bytes()
```

Denne kommandoen tar i mot teksten som blir sendt og legger den i en variabel kalt x.

Musikk
- med Python Mu

En micro:bit kan enkelt kobles til en summer (også kalt buzzer) eller høyttaler for å spille av lyd. Siden en summer bare kan spille av en enkelt tone, vil det bli mye finere lyd fra en høyttaler, da den kan spille de overtonene vi er vant med å høre som en del av de rene tonene i musikk.

For å kunne bruke en høyttaler eller en summer i Python Mu, må vi importere et ferdig bibliotek som inneholder metoder for å lage og kontrollere lyd.

```
from music import *
```

Når vi skal spille av en tone i Python Mu, må vi oppgi tre ting som tekst (strenger). Tonens navn, hvilken oktav den tilhører og hvor lang tid den skal spilles. Et piano dekker om lag syv oktaver, MakeCode for micro:bit dekker tre oktaver og Python Mu dekker ni oktaver. Oktav 0 er den laveste oktaven, som gir de mørkeste tonene. Oktav nummer fire er den som inneholder kammertonen A med frekvens 440 Hz. I MakeCode tilsvarer lav oktav (liten) oktav nummer tre, midtre oktav (enstrøken) oktav nummer fire, og høy oktav (tostrøken) oktav nummer fem i Python Mu. Lengden på tonen oppgis som et tall som tilsvarer antall slag multiplisert med 4. Der lengden på hvert slag er definert av Mu. Funksjonen under spiller av fire toner.

```
music.play("C3:4", "D3:4", "E3:4", "C#3:8")
```

Navn på tonen

Hvilken oktav tonen tilhører

Hvor mange slag tonen skal spilles

Dersom vi ønsker å skrive litt mindre, kan play()-funksjonen huske hvilken oktav vi bruker og hvor mange slag tonen varer. Dette er spesielt nyttig å bruke når vi skriver hele melodier, da slipper det å bli fullt så mye å skrive. Som regel vil en melodi i hovedsak være del av bare én oktav. Når vi benytter oss av Mus hukommelse, ser kommandoen over slik ut:

```
music.play("C3:4", "D", "E", "C#:8")
```

Dersom du vil bestemme selv akkurat hvor lenge hver tone skal spilles, kan du bruke en funksjon som kalles pitch(). Da må du oppgi frekvensen til tonen, og ikke navnet på den. Noen frekvenser er oppgitt i tabellen i opplegg 11. Spilletiden oppgis i millisekunder, og dermed kan du bestemme tempoet (bpm) med stor nøyaktighet.

```
music.pitch(440, 1000, pin1)
```

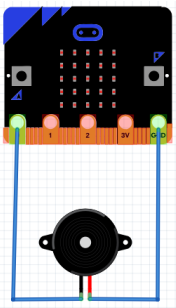
Frekvensen

Varighet i millisekunder

Hvilken utgang høyttaleren (summeren) er koblet til. Denne trenger du ikke skrive noe på, dersom du bruker pin0, da den ligger inne som standardverdi.

Det finnes også en rekke ferdige melodier. Du kan finne dem på denne siden: https://microbit-micropython.readthedocs.io/en/v1.0.1/tutorials/music.html

Det har ikke noe å si hvilken utgang på høyttaleren eller summeren du kobler til hvilken utgang på micro:biten.



Oppgave

Lag et program som spiller av «Lisa gikk til skolen».

Lag en egen melodi som du spiller av på micro:biten.

Logge til PC/MAC med Python Mu

Noe det kan være veldig fint å bruke micro:bit til, er som en datalogger. Det vil si at vi bruker den til å samle inn data, slik vi har gjort mye av, også får vi datamaskinen til å skrive opp målingene på skjermen og lage en graf av dem mens vi måler.

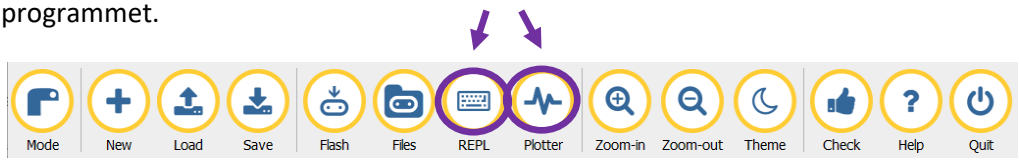
For å vise noe på skjermen bruker vi print()-kommandoen. Det som står inni parentesen er det som kommandoen skriver til skjermen, altså viser på skjermen. Dersom vi bare er interessert i å få ut en liste med målinger etter hvert som de blir målt, kan vi bare bruke en kodesnutt som ser slik ut:

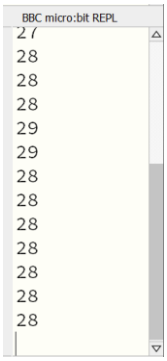
```
x = temperature()
print(x)
```

Måler temperaturen og legger verdien i variabel x.

Skriver ut verdien av variabel x til skjermen.

Etter at vi har overført hele programmet til micro:biten, må vi trykke på to knapper for å få fram tabellen med målingene. Trykk først på «Plotter», deretter på «REPL». Det er viktig å gjøre dette i denne rekkefølgen, selv om vi ikke skal vise noen graf (plotteren viser grafen), ellers krasjer programmet.





Slik ser da listen med målingene ut.

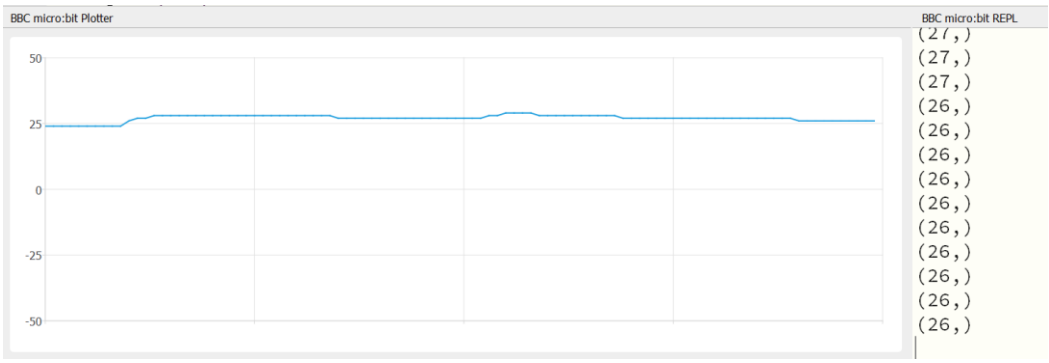
Ofte vil vi heller se grafen for målingene, og da må vi forandre litt på koden vår. I Python Mu vil vi få vist grafer, dersom vi har riktig format på det vi skriver ut til skjermen. Dette formatet blir vist under.

```
x = temperature()
print((x,))
```

Diskuter

Hva er forskjellen på de to kodesnuttene på denne siden?

Når det nye programmet er overført, og du åpner plotter, deretter REPL, så vil vi se både grafen og en liste over verdiene. Denne lista er litt mer tungvint å lese ut fra, men så lenge man klarer å se bort fra ekstra parenteser og komma, så står verdiene fra temperaturmålingene som på figuren under.



Oppgave

Lag hele programmet som måler temperaturen og plotter den som en graf på skjermen.

Hva skjer om du skriver inn en annen verdi i stedet for et tomt felt i print()-kommandoen?

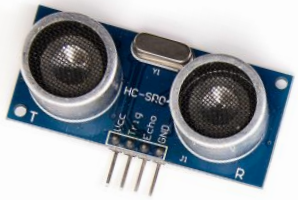
Diskuter

Å logge data kan vi gjøre med alle de innebyggede sensorene til micro:bit, men også med alle sensorer vi kobler på selv.

Hva må du endre i koden for å få til dette?

Avstandsmåler (sonar)

- med Python Mu

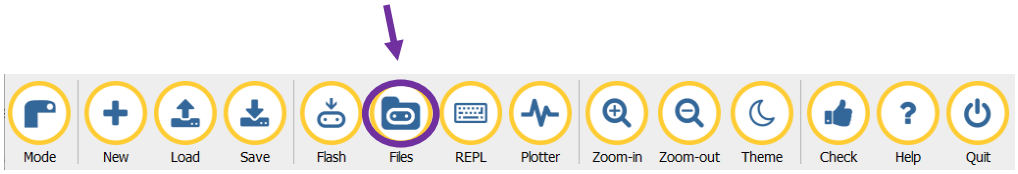


En sonar sender ut et lydsignal, og måler hvor lang tid det tar før det signalet blir reflektert tilbake til sonaren. Dersom man vet hva lydfarten er, så kan vi enkelt regne ut avstanden til det som reflekterer lydsignalet. Sonaren vet hva lydfarten er, og vi kan dermed bruke den til å måle avstander med. For å kunne bruke en sonar i Python Mu, så må vi importere et bibliotek på selve micro:biten. Dette biblioteket er egentlig et program som gjør det enklere for oss å programmere servoen, uten å måtte tenke på hvordan alle detaljene i målingene.

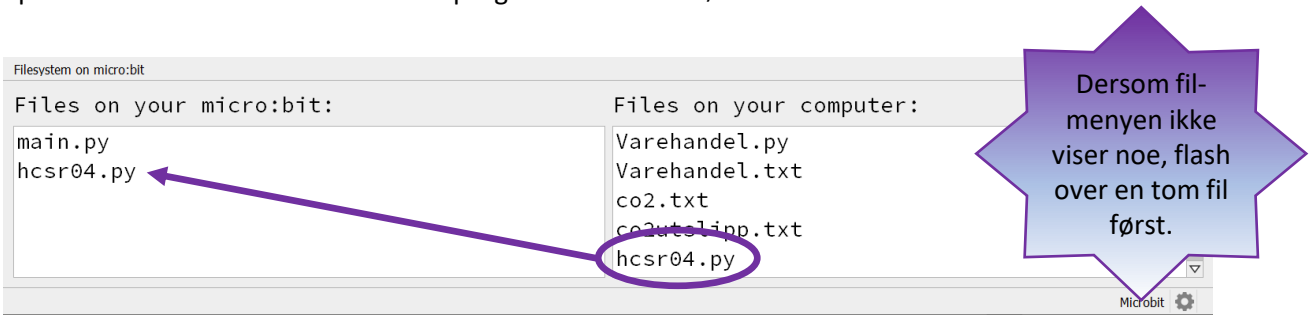
Det første vi må gjøre er å få tak i biblioteket til sonaren, og det kan finnes på www.kunnskapsfilm.no\.....

Kopier koden som står på siden i en ny fane (fil) i Python Mu og lagre den som «hcsr04.py» ved å trykke på «Save» og lagre den i katalogen som dukker direkte opp.

Så må denne fila legges over på selve micro:biten, og det gjøres ved å koble micro:biten til datamaskinen, og trykke på «Files».



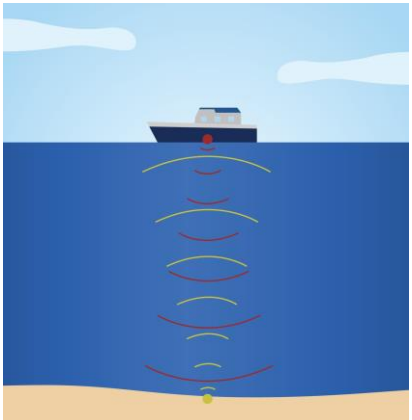
Da kommer denne menyen opp i Mu, og du må lete i feltet på høyre side til du finner fila som du nettopp lagret: «hcsr04.py». Når du finner den, drar du den over i det venstre feltet. Da vil lyset på baksiden av micro:biten blinke til programmet er overført.



Da gjenstår det å lage selve programmet som bruker sonaren til å måle avstander med. Under er det en puslespill-oppgave der kodelinjene må ryddes i riktig rekkefølge for å lage dette programmet.

Puslespill-oppgave

```
print((HCSR04(),))
from hcsr04 import HCSR04
sleep(1000)
while True:
from microbit import *
```



Servo
- med Python Mu

En servo mottar et signal fra micro:biten slik at den beveger seg slik vi ønsker. Det finnes to typer servoer, 180° og kontinuerlige (360°), og de blir styrt på litt forskjellig måte. 180°-servoen kan bevege seg fra en -90°-vinkel til en 90°-vinkel, altså 180° til sammen. For en kontinuerlig servo oppgir vi hvor raskt den skal snurre rundt i prosent. Selve hastigheten kan variere litt med hvor mye strøm som er igjen i batteriene, for servoer trekker ganske mye strøm. Så dersom du har en servo på 100 % hastighet på begynnelsen og slutten av skoletimen, vil den snurre en del saktere på slutten av timen.

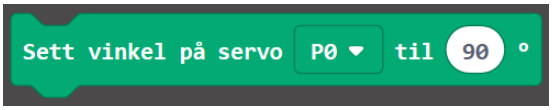
For å kunne bruke en servo i Python Mu, må vi importere et bibliotek på selve micro:biten. Dette biblioteket er egentlig et program som gjør det enklere for oss å programmere servoen, uten å måtte tenke på alle detaljene i signalene som styrer den, tilsvarende som for sonaren.

Det første vi må gjøre er å få tak i biblioteket til servoen, og det og det kan finnes på nettressursen til Kobling på www.kunnskapsfilm.no.

Kopier koden som står på siden i en ny fane (fil) i Python Mu og lagre den som «servo.py» ved å trykke på «Save» og lagre den i katalogen som dukker direkte opp.

Så må denne fila legges over på selve micro:biten, og det gjøres slik som for sonaren på forrige side.

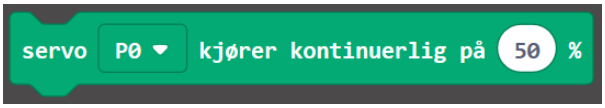
180°-servo




```
servovinkel(pin0, 90)
```

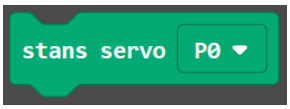
Ved å bruke denne kommandoen, kan vi stille in hvilken vinkel servoen skal bevege seg til. Kommandoen tar inn hvilken utgang vi har koblet servoen til, og hvilken vinkel servoen skal bevege seg til. Vinkelen kan variere mellom -90° og 90°.

Kontinuerlig servo (360°)




```
servohastighet(pin0, 50)
```

Ved å bruke denne kommandoen, kan vi stille in hvilken hastighet servoen skal snurre med. Denne kommandoen tar også inn hvilken utgang vi har koblet servoen til, i tillegg til hastigheten i prosent. Hastigheten kan variere mellom -100 % og 100 %, der fortegnet bestemmer i hvilken retning servoen skal snurre.




```
servostopp(pin0)
```

Denne kommandoen får en kontinuerlig servo til å slutte å snurre rundt. Det går også an å bruke kommandoen servohastighet() til dette.

Skrive til fil på micro:bit

- med Python Mu

I slutten av kapittel 9 brukte vi datasett i form av tekstfiler for å lage matematiske modeller. Hadde det ikke vært kult om vi kunne fått micro:biten til å lage slike filer av sine målinger? Da kan vi bruke micro:biten til å samle inn data over en viss periode, uten å være koblet til en datamaskin. Datasettet blir lagret i micro:biten som en .txt-fil, og vi kan legge den over i datamaskinen, før vi plotter resultatene eller lager en matematisk modell med Python.

Det aller første vi må gjøre, er å opprette en tekstfil, det vil i en fil som slutter på .txt, og det er enklest å gjøre i Mu. Lagre den samme i samme mappe som resten av dine Python Mu-filer.

Deretter må den fila du har laget, kopieres over på micro:biten. Dette gjør du på samme måten som for sonaren.

```
with open ('filnavn.txt', 'w') as fil:
```

Åpner fila som heter filnavn.txt og tildeler den til et objekt som vi kaller fil. Siden vi har 'w' (write) i kommandoen gjøres det klar til å skrive inn i fila.

```
with open ('filnavn.txt', 'r') as fil:
```

Tilsvarende som linja over, men siden vi har 'r' (read) i kommandoen gjøres det klar til å lese fila. Det er valgfritt å ta med 'r'.

```
fil.write(y+'\n')
```

Skriver variabelen y til fila vår. +'n' gjør at det blir et linjeskift etter hvert tall, og det må vi ha for at python skal kunne lese filene. Må stå med innrykk.

```
y = str(x)
```

Gjør om variabelen x til en streng (tekstverdi). Må stå med innrykk.

```
z = fil.read()
```

Leser det som står i fila vår, og legger det inn i variabel z. Må stå med innrykk.

```
print(z)
```

Skriver variabelen z til skjermen på PC. Må IKKE stå med innrykk.

Da gjenstår det å lage selve programmet. Under er et eksempel der micro:biten måler temperaturen og legger målingen i en .txt-fil ved navn datasett.txt. Det kan være greit å kunne sjekke hvordan datasett.txt-fila ser ut uten å overføre til datamaskinen og åpne den, derfor er dette også med i puslespill-oppgaven.

Puslespill-oppgave

```
print(innhold)
```

```
while not button_a.is_pressed():
```

```
sleep(1000)
```

```
with open('datasett.txt') as fil:
```

```
with open('datasett.txt', 'w') as fil:
```

```
innhold = fil.read()
```

```
t = temperature()
```

```
fil.write(str(t)+'\n')
```

```
from microbit import *
```

Ekstraoppgave

Lag et program som bruker en sensor du kobler på micro:biten for å samle inn data som du lagrer i en fil på micro:biten. Hva må du forandre på?

Hva må du forandre i programmet for å lage en fil med to kolonner der den første kolonnen er hvor lang tid det har gått før målingen er gjort?