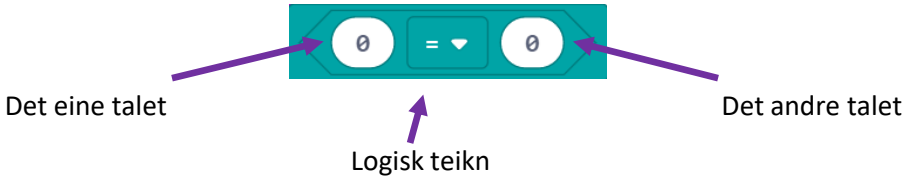
Introduksjon til Python-programmering

Vilkår (conditions)

Vilkår i Python er det same som vilkår i Scratch eller MakeCode. Skilnaden mellom dei er at i staden for å ha ferdige sekskanta klossar med vilkår, må vi skrive vilkåra sjølv i Python. Eit vilkår er ei utsegn som kan vere sann eller usann, og den typen vilkår vi kan bruke i programmering vil alltid samanlikne noko. Ofte samanliknar vi talverdiar og ser om dei er like store, om det første er mindre enn det andre, eller motsett. Det går og an å samanlikne tekstar for å sjå om dei er like eller ikkje. Men for tekstar kan ein ikkje finne ut om den eine teksten er større enn ein anna. Korleis skulle vi målt det?

Eit vilkår i python må vere samansett av tre delar, akkurat som i Scratch og MakeCode. To tal som skal samanliknast på ein eller annan måte, og eit teikn som seier korleis dei skal samanliknast. Dette teiknet kallast ofte for eit logisk teikn eller ein logisk operator. Dersom sammenlikninga stemmer, vil vilkåret gje oss verdien «sann» attende. Dersom samanlikninga ikkje stemmer, vil vi i staden få verdien «usann» attende. Sann og usann kallast boolske verdiar. Det går an å ha boolske variablar, det vil sei at variablen kan lagre verdien sann eller usann.



Under kjem eit oversyn over korleis ein kan skrive ulike vilkår. Sjå om du klarer å finne ut om døma returnerer verdien sann eller usann.

Vilkår		
Teikn	**Forklaring**	**Døme**
>	Større enn	4 > 5
<	Mindre enn	2 < 7
>=	Større enn eller lik	-2 >= 0
<=	Mindre enn eller lik	7 <= 12
==	Lik	9 == 19
!=	Ikkje lik	3 != 6
or	Eitt av vilkåra må vere sann	3 == 4 or 3 > 1
and	Alle vilkåra må vere sanne	3 == 4 and 3 > 1

Rekning

Nokre gonger treng vi å gjere reknestykke i Python. Dette er særs nyttig der vi har nokre variablar som vi, til dømes, skal multiplisere eller legge saman. Men det går fint an å gjere reknestykke direkte med tal. Under er eit oversyn over ulike rekneoperatorar for Python. Sjå om du klarer å sjå kva verdien av reknestykke-døma blir.

Rekneoperatorar		
Teikn	**Forklaring**	**Døme**
+	Addisjon	(-4) + 5
-	Subtraksjon	(-1) - (-4)
*	Multiplikasjon	3 * 0
/	Divisjon	12 / (-3)
**	Opphøgd i (eksponent)	2**3

Variablar i Python

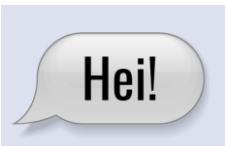
Ulike typar variablar er viktige innan programmering. Eitt av poenga med å lage program, er at vi skal kunne bruke det same programmet på å løyse fleire liknande oppgåver. Da er det mykje greiare å lagre alle tal vi skal bruke i variablar, slik at vi berre treng byte ut tala ein stad, for at programmet likevel skal gi oss rett svar.

Ein variabel kan romme éin verdi, og kvar gong vi legg ein verdi i ein variabel, erstattar vi den verdien som eventuelt ligg i variabelen frå før. Når vi lagar ein variabel i Python, treng vi ikkje skrive kva type det er, det kjenner Python att ut frå den første verdien vi legg inn i variabelen. Det finnest fire typar variablar:

Heiltal
- integer

Desimaltal
- float

Tekst
- string



Boolsk verdi
- boolean

Heiltal kallast integer i Python. Ein integer er eit positivt eller negativt tall som ikkje har nokon desimalar. 0 er og ein integer.

Desimaltal kallast float i Python. Ein float er eit tal som inneheld desimalar. Det kan vere enten negativt eller positivt. Talet 2,0 er ein float.

Tekst kallast string i Python. (På norsk blir og ordet «streng» brukt.) Ein tekstverdi er eitt eller fleire ord, det kan og vere heile setningar. For at python skal skjøne at det er ein tekstverdi, må vi bruke apostrofar (‘) rundt teksten. Dersom vi vil legge ordet hei inn i variabelen a, må vi skrive kommandoen slik: `a = 'hei'`

Boolsk verdi kallast boolean value i Python. Denne verdien kan vere enten sann eller usann. Han blir brukt i samband med vilkår, sidan eit vilkår kan vere enten sant eller usant.

Av og til treng vi å gjere om ein type variabel til ein annan type variabel. Slik som når det er en funksjon i Python som berre tek inn heiltall. Dersom vi da har eit desimaltal, må vi gjere det om til eit heiltal, og da må vi avrunde desimaltalet vårt. Da bruker vi kommandoen round(). Det går og an å bruke kommandoen int(), men denne avrunder ikkje desimaltalet, han berre kuttar ut alt bak kommaet. Det tyder i praksis at desimaltalet alltid blir runda ned til nærmaste heiltal.

For å kunne sende tal mellom to eller fleire micro:bitar, må verdiane vere gjort om til tekstverdiar. For å gjere om ein talverdi til ein tekstverd,i kan vi bruke kommandoen str().

Oppgåve
Kva vil bli vist på skjermen for desse programma? Kan du lage nokre program sjølv? Få ein i klassen til å gisse kva som vil skje i programma dine, eller kan hende læraren?

```
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)
```

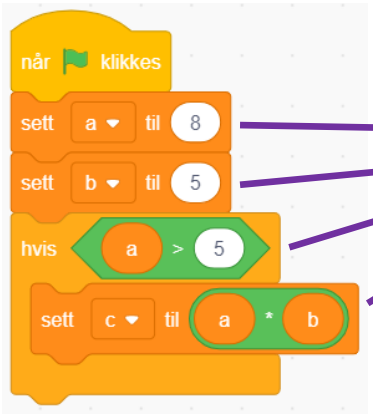
```
a = '1.7'
a = float(a)
print(a)
```

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

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

Viss-setningar (if-setningar)

Viss-setningar svarer til viss-klossar i Scratch og MakeCode. Dei må romme eit krav, og dersom kravet er sant, vil det som «høyrer til» viss-setningen bli utført. I dei klossbaserte programmeringsspråka puttar vi inn det som skal skje med hjelp av klossane som passar slik som puslespelbrikker. I Pyton må vi gjere det same, berre at vi må skrive alt sjølv. Sidan vi nå ikkje har klossar for å vise kva som står «inni» viss-setningen, bruker vi innrykk i staden. Alt som står etter ein dersom-setning, med eit innrykk framføre, vil «stå inni» viss-setningen, og blir berre gjort om vilkåret i viss-setningen er sant.




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

Desse to programma gjer det same, og kvar blokk i Scratch tilsvarer éi linje i Python, med to unnatak.

I Python treng vi ikkje å trykke på eit flagg, vi trykker på ein «play-knapp» som får koden til å byrje å køyre. Dermed treng vi ikkje noko som svarer til den første blokka i Scratch-koden.

I Python viser ikkje verdien på variablane, med mindre vi skriv dei ut til skjermen med å bruke print-kommandoen. I Scratch visest verdien til alle variablane heile tida, så der treng vi ikkje å få vist fram c på nokon spesiell måte.

Ofte treng vi å ha fleire alternativ som kan vere sanne. Det finnest to ulike variantar, og dei kallest gjerne «elles» eller «elles hvis».

Elles – det som skal skje, om vilkåret i dersom-setningen er usant.

```
else:
```

Elles hvis – det som skal skje om eit anna vilkår er sant.

```
elif:
```

Snakk om

Kva måtte du endre i Python-programmet for at det berre skulle rekne ut c når verdien i b var større enn verdien i a?

Kva for ein skilnad hadde det blitt dersom vi flytta linja med print(c) heilt til venstre i Pythonkoden, slik at ho ikkje sto med innrykk?

Oppgåve

Kva vil bli vist på skjermen for desse programma? Kan du lage nokre program sjølv? Få ein i klassen til å gisse kva som vil skje i programma dine, eller kan hende læraren?

```
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)
```

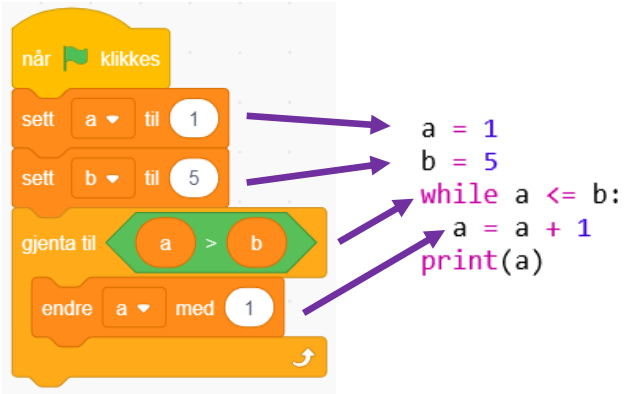
```
a = 5
b = 8
if a > b:
    c = a * b
    print(c)
elif a==b:
    print(a)
elif a < b:
    print(a)
    print(b)
```

Korleis skrive viss-setningar – ei oppskrift

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

While-lykkje i Python

While-lykkjer (vilkårlykkjer) svarer til «gjenta til-lykkjene» i Scratch. Dei må ha med eit vilkår som fortel kor mange gonger det som står inni lykkja skal gjerast. For Scratch er det slik at det som står inni lykkja blir gjort heilt til vilkåret blir sant (altså at lykkja går medan vilkåret er usant). I Python er det motsett, at det som står inni lykkja blir gjort heilt til vilkåret ikkje lenger er sant. Det gjer at sjølve vilkåret må skrivast på ulike måtar dersom ein skal oppnå det same med Scratch og Python.



Desse to programma gjer det same, og kvar blokk i Scratch svarer til éi linje i Python, men med to unnatak.

I Python treng vi ikkje trykke på eit flagg, vi trykker på ein «play-knapp» som får koden til å byrje å køyre. Dermed treng vi ikkje noko som svarer til den første klossen i Scratch-koden.

I Python viser ikje verdien på variablane, med mindre vi skriv dei ut til skjermen med å bruke print-kommandoen. I Scratch visest verdien til alle variablane heile tida, så der treng vi ikkje å få vist fram a på nokon særs måte, men vi rekk ikkje å sjå kva verdiar a har undervegs, sidan programmet går så raskt.

Snakk om

Kva vil vise på skjermen for Python-programmet?

Vilkåret i dei to programma er ikkje like. Kva ville Pythonprogrammet vist dersom ein brukte vilkåret frå Scratch-programmet direkte?

Kva for ein verdi ville a hatt, dersom vi brukte vilkåret frå Pythonprogrammet direkte i Scratch-programmet?

Korleis skrive while-lykkjer – ei oppskrift

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

Pass på å sette alt som skal vere inni while-lykkja med innrykk. Dersom det ikkje står med innrykk, vil det ikke skje før lykkja er ferdig, og programmet held fram.

Oppgåve

1. Hva vil bli vist på skjermen for desse programma?

Skriv ned kva du trur vil bli vist på skjermen, og skriv inn programma med Pythonkode etterpå. Køyr koden, og sjekk om du får det svaret du trudde du skulle få.

2. Dersom du ikkje fikk svaret du trudde, kva var grunnen til det, trur du? Diskuter gjerne med andre.
3. Kan du lage nokre program sjølv? Få ein i klassen til å gisse kva som vil skje i programma dine, eller kan hende læraren?

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

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

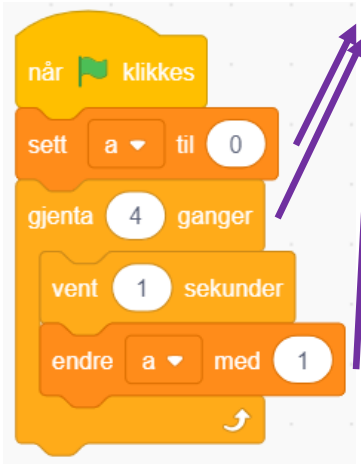
```
a = 10
b = 0
while a > b:
  a = a - 1
  print(a)
```

```
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-lykkjer i Python

For-lykkjer (teljelykkjer) svarer til «repeter eit visst tal gonger-lykkjene» i Scratch. Desse lykkjene gjentek det som står inni dei så mange gonger som vi ønsker. I Scratch kan vi berre skrive direkte kor mange gonger vi vil gjenta lykkja, men i Python er det litt annleis. Der blir det oppretta ein variabel som heiter i, vi kan gjerne kalle det ein teljevariabel. Han tek til på den talverdien som står først i parentesen til for-lykkja, i dømet vårt er det 0. Det andre talet i parentesen viser kva tal teljervariabelen i skal telje seg opp til (men ikkje inkludert). I situasjonen vår skal i telje seg opp til verdien 5. Det siste talet inni parentesen seier kor store steg vi skal auke verdien til teljevariabelen i med. I dømet vårt skal vi telje med 1, altså seier vi at teljervariabelen tek til på 0, vi tel opp til 5 med å auke med 1 kvar gong. Da vil i vere 0, 1, 2, 3, 4 før lykkja blir avslutta.




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

Snakk om

Kva ville skjedd om vi brukte variabel a i staden for talet 4 i lykkja i Scratch?

Går både Scratch-programmet og Pythonprogrammet gjennom lykkja like mange gonger?

Korleis skrive for-lykkjer – ei oppskrift

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

Desse to programma gjer det same, men her svarer ikkje kvar kloss i Scratch til éi linje i Python. Her er tre av blokkene i Scratch innbakte i for-lykkja i python. I tillegg er det to klossar i Scratch som ikkje svarer til nokon linjer i Python.

I Python treng vi ikkje trykke på eit flagg, vi trykker på ein «play-knapp» som får koden til å byrje å køyre. Dermed treng vi ikkje noko som svarer til den første klossen i Scratch-koden.

I Python viser ikkje verdien på variablane med mindre vi skriv dei ut til skjermen med å bruke print-kommandoen. I Scratch visest verdien til alle variablane heile tida, så der treng vi ikkje å få vist fram a på nokon særs måte, men vi rekk ikkje å sjå kva verdiar han har undervegs sidan det går så raskt. For å kunne sjå kva for nokre verdiar variabelen i Scratch har, har vi denne gongen lagt inn ein kloss som seier «vent 1 sekund», og denne gjer at programmet tek ein pause i eitt sekund. Da rekk vi å sjå verdien på variabelen før programmet går vidare att.

Oppgåve

1. Kva vil bli vist på skjermen for desse fem programma?

Skriv ned kva du trur vil visest på skjermen, og skriv inn programma med Pythonkode etterpå.
Køyr koden, og sjekk om du får det svaret du trudde du skulle få.

2. Dersom du ikkje fekk svaret du trudde, hva var grunnen til det, trur du? Diskuter gjerne med andre.
3. Kan du lage nokre program sjølv? Få ein i klassen til å gisse kva som vil skje i programma dine, eller kan hende læraren?

```
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 treng vi å bruke mange variablar, og da kan det vere greit å lage ei liste eller ein tabell (array i Python). Dersom éin variabel svarer til éin skuff som har eit namn og ein verdi, er ei liste som ein kommode med fleire skuffar. Kvar variabel er eitt element i lista, og kvart element har eit namn og ein verdi. Namnet på elementet er namnet på heile lista saman med plasseringa på lista, altså kva for ei rad det høyrer til.

Dermed treng vi ikkje å lage et eige namn til kvart element. Det er og enkelt å bruke ei lykkje for å gå gjennom alle elementa for å bruke dei, til dømes i eit reknestykke.

Ei liste har eit namn og element med kvart sitt nummer. Det som er litt spesielt med lister innan programmering, er at det første elementet er element nummer 0. Det er fort gjort å gløyme når ein programmerer, og er alltid lurt å sjekke som ein del av feilsøkinga. Ei liste har i utgangspunktet berre element med éin datatype. Det går ikkje an å bruke nokon heiltal og nokon tekstverdiar i same liste. Ei liste er i utgangspunktet ei lang rekke med element, og er med det eindimensjonal.

Namnet på lista må vere eitt samanhengande ord for at Python skal skjøne det. Innan programmering er det vanleg å bruke _ som mellomrom for namn på variablar, lister og tabellar. Bokstavane æ, ø og å bør vi unngå, sidan nokre Python-editorar ikkje taklar namn som bruker andre enn engelske teikn.

Ønskeliste

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

Eit element har både et namn og ein verdi.

Kva er namnet og kva er verdien til dette elementet?

Oversyn over nokre nyttige kommandoar i Python

```
liste = []
```

Lagar ei tom liste.

```
liste = [1, 3, 5, 7]
```

Lagar ei liste med verdiane 1, 3, 5, 7.

```
liste[3] = 7
```

Legg inn verdien 7 i element tre, i ei liste med minst fire element.

```
del liste[3]
```

Slettar element tre, i ei liste med minst fire element.

```
liste.append(4)
```

Legg til verdien fire som eit nytt element i ei liste som heiter liste. Det har ikkje noko å seie kor mange element som er i lista frå før, for verdien blir alltid lagt i eit nytt element etter det siste elementet i den opprinnelege lista.

```
print(liste[2])
```

Hentar verdien til element to i lista, og skriv det på skjermen.

```
len(liste)
```

Finn kor mange element det er i ei liste.

Diskusjonsoppgåver

1. Kvifor er det liten vits i å bruke ei lykkje for å legge saman mange variablar?
2. Kvifor er det enkelt å bruke ei lykkje for å gjere det same dersom du har alle verdiane i ei liste?

Tabellar i Python

Ein tabell (array) kan innehalde fleire typer data, og han kan vere todimensjonal. Det vil seie at han har meir enn éin kolonne med data. Da treng vi to tal for å forklare kva for eit element i tabellen vi snakkar om. Det blir på same måten som eit punkt i eit koordinatsystem, der vi må bruke både x- og y- koordinaten for å plassere eit punkt. Det første elementet tek til på 0 for tabellar og, både for radene og for kolonnane. Tabellar finnest i numpy-biblioteket, og må importerast `from numpy import*`

Reine tekstfiler (.txt-filer) frå statistisk sentralbyrå gir todimensjonale (2D) tabellar i Python, og er med det viktige når vi skal analysere data og lage matematiske modellar. Det er ganske enkelt å gjere om ein 2D- array til ein 1D-array som er det fit_function() tek inn av data (sjå kapittel 13).

Tabellen under har namnet budsjett. Akkurat som for lister, byrjar vi å telle på null. Dette gjeld for både radene (vassrett) og kolonnane (loddrett). Rad null i budsjett-tabellen inneheld tekstverdiane: Månad, Inntekt, Klede og snacks, Mobilutgifter, Diverse utgifter, SUM. Kolonne null inneheld Månad, Januar, Februar, Mars, April, Mai, Juni.

Månad	Inntekt	Klede 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.

Diskusjonsoppgåver

1. Kva for ein verdi har element [0,0]?
2. Kva for eit element har verdien 3000?

Oversyn over nokre nyttige kommandoar

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

Endrar verdien frå 1500 til 2000 i elementet i rad sju og kolonne to i budsjett-tabellen på denne sida.

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

Skriv ut verdien til elementet i rad tre og kolonne ein, altså 1500 frå budsjettet på denne sida.

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

Lagar ein tabell med to rader og tre kolonnar, der alle verdiene er null.

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

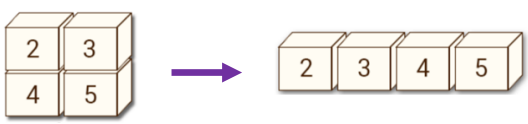
Lagar ein tabell med to rader og tre kolonnar. Tala innanfor parentesane gjev verdiane til elementa i kvar rad.

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

Lagar ein 1D-tabell med 11 verdiar jamnt fordelt frå og med null til og med fem.

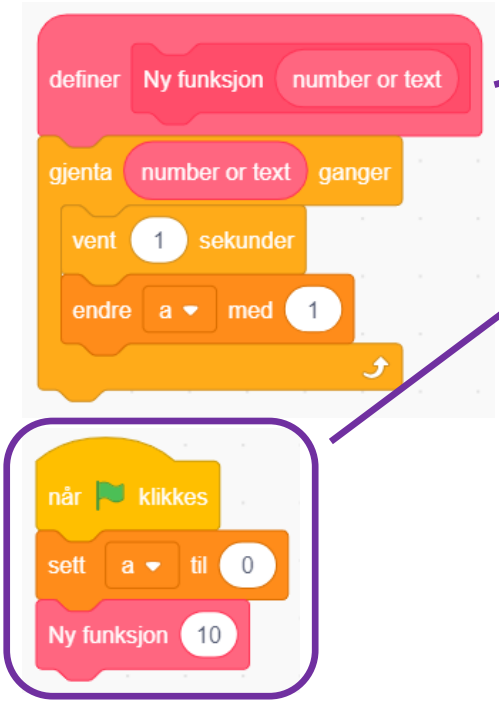
```
a = tabell.flatten()
```

Gjer om en 2D-tabell til ein 1D-tabell. Alle radene i 2D-tabellen blir lagt etter kvarandre til ei lang rad.



Funksjonar i Python

Funksjonar i Python svarer til «lag ein ny kloss» i Scratch, og er eigentleg berre småprogram. Faktisk treng ein funksjon ikkje vere veldig liten heller, men det som skil han frå eit vanleg program, er at han må bli definert før han kan brukast. Når funksjonen er definert, kan vi bruke han så mange gonger vi ønsker utan å skrive all koden på ny, berre med å skrive éi kodelinje.




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

Funksjon(10)
```

Desse to programma gjer det same, men her svarar ikkje kvar kloss i Scratch til éi linje i Python. Funksjonen som blir definert, inneheld ei for-lykkje som ikkje er bygd opp på heilt lik måte i Scratch og Python (se for-lykkje-sida).

I Python treng vi ikkje trykke på eit flagg, men vi må legge inn ei linje for å vise variabelen på skjermen. For å kunne sjå alle verdiane på skjermen, har vi lagt inn ein pause på eitt sekund i Scratch-programmet.

Både i Scratch og Python kan funksjonane ta imot inn-verdiar. Det vil seie at når vi bruker funksjonen, må vi oppgje éin eller fleire verdiar som blir plasserte i variablar. Desse verdiane blir da brukt til noko i funksjonen, gjerne ei utrekning. I dømet heiter variabelen til inn-verdien i Pythonprogrammet inndata.

Ein viktig skilnad mellom funksjonar i Scratch og Python, er at i Python kan funksjonane gi oss ein ut-verdi. Det vil seie at vi til dømes kan bruke ein funksjon til å rekne ut eit reknestykke for oss, og gi oss eit tal attende. Dette talet kan vi til dømes sette inn i nye reknestykke, i viss-setningar eller i lykkjer. I Scratch går ikkje dette an.

Snakk om

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

Går både Scratch-programmet og Pythonprogrammet gjennom lykkja like mange gonger?

Oppgåve

1. Kva vil bli vist på skjermen for desse tre programma?

Skriv ned kva du trur vil visast på skjermen, og skriv inn programma med Pythonkode etterpå.
Køyr koden, og sjekk om du får det svaret du trudde du skulle få.

2. Dersom du ikkje fekk svaret du trudde, kva var grunnen til det, trur du? Diskuter gjerne med andre.
3. Kan du lage nokre program sjølv? Få ein i klassen til å gisse kva som vil skje i programma dine, eller kan hende læraren?

```
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))
```

Korleis definere og bruke funksjonar – ei oppskrift

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

navn(variabler)
```

Teikning i Python - turtle

Python har eit bibliotek som heiter turtle, som kan brukast når ein ønsker å bruke Python til å teikne figurar eller mønster. For å importere dette biblioteket må ein skrive: `from turtle import *` øverst i programmet.

Døme på nokre grunnleggande funksjonar frå turtle-biblioteket

```
t = Turtle()
```

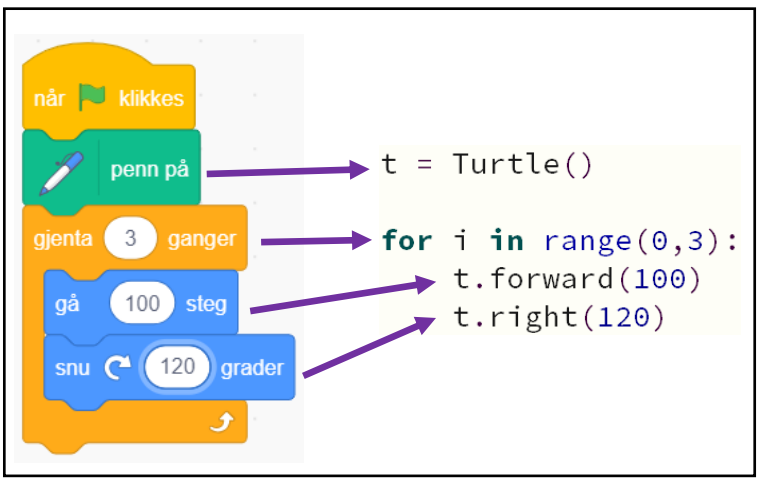
Denne funksjonen lagar eit «turtle-objekt» som ein treng når ein bruker dei andre funksjonane som lagar teikninga. Vi kan tenke på det som ein blyant eller penn.

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

Desse funksjonane flytter turtle fram eller attende 100 steg.

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

Desse funksjonane snur turtle 120 grader til venstre eller til høgre (det er det same som at den ytre vinkelen er 60 grader.




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

Denne funksjonen teikner ein sirkel med radius 50 steg.

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

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

Oppgåve

1. Kva vil visast på skjermen for desse programma?

Skriv ned kva du trur vil visast på skjermen, og skriv inn programma med Pythonkode etterpå. Køyr koden, og sjekk om du får det svaret du trudde du skulle få.

2. Dersom du ikkje fekk svaret du trudde, kva var grunnen til det, trur du? Diskuter gjerne med andre.

3. Kan du lage nokre program sjølv? Få ein i klassen til å gisse kva som blir teikna med programma dine, eller kan hende læraren?

```
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)
```

Oversyn over nokre kommandoar for micro:bit i Python Mu

Når vi skal bruke Python for å programmere ein micro:bit, finst det ein del ferdige kommandoar vi kan nytte. Her viser vi eit oversynt over nokre av de mest nyttige. Andre døme kan du finne på denne nettsida: https://microbit-micropython.readthedocs.io/en/v1.0.1/tutorials/introduction.html

Vi må alltid ta til med å importere det biblioteket som inneheld micro:bit-kommandoane. Det gjer ein ved å skrive øvst:

```
from microbit import *
```

Inndata

Knappar

Sidan micro:biten har to knappar, A og B, kan den delen av koden der det står button_a i døma bytast ut med button_b i staden.

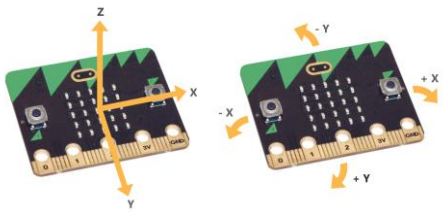
```
button_a.is_pressed()
```


```
button_a.get_presses()
```

Denne kommandoen tel talet på gonger ein knapp har vore trykt sidan programmet starta.

Innebygde sensorar

Vi kan bruke micro:bitens innebygde sensorar med Python-programmering og. For akselerasjonen kan denne målast i alle dei tre romlige retningane vi har. Tenk på det som eit koordinatsystem med 3 aksar. Dei vanlege, x-aksen og y-aksen, men og ein z-akse som står loddrett på begge dei to andre aksane.



```
accelerometer.get_x()
```

```
temperature()
```

```
display.read_light_level()
```

```
running_time()
```

Påkopla sensorar – sjå og sida om sonar

Ein digital verdi kan berre vere enten 1 eller 0, der 1 tyder på og 0 tyder av. Den digitale verdien 1 er eigentleg ei høg spenning, medan den digitale verdien 0 er ei veldig låg spenning. Digitale verdiar bruker vi sjelden for sensorar, berre dersom vi skal sjekke om noko er avskrudd eller påskrudd. Sensorar vi koplar på micro:biten gir som regel verdiar mellom 0 og 1023, og da må vi lese av ein analog verdi i staden. Ein analog verdi kan vere mykje anna enn berre 0 eller 1, og passar med det best for å lese av kva for ein verdi sensoren gir.

```
pin0.read_analog()
```


```
pin0.read_digital()
```

Utdata

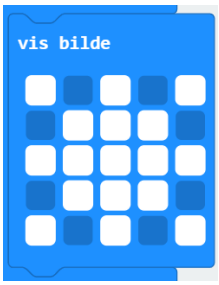
Skrive til skjermen

Det går an å skru på dei einskilde lysdiodane på skjermen til micro:biten, med verdiar mellom 0 og 9 for kor sterkt dei skal lyse.

```
display.show()
```


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


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



Påkopla lysdiodar

Når du koplar på ein lysdiode til micro:biten, kan du programmere han til å skru seg av eller på. Da må ein bruke kommandoen som sender ut digital data, altså verdien 1 eller 0, der 1 svarer til på og 0 svarer til av.

Kva for ein utgang lysdioden er tilkopla (0, 1 eller 2).

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

På (1) eller av (0).

Sende data over radio-forbindelsen

På same måten som med å bruke klossar, kan vi programmere micro:bitar til å sende data til kvarandre med å bruke Python.

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

Import radio-kommandoen importerer biblioteket som gjer at vi kan bruke radio-sambandet mellom micro:bitane. I tillegg må vi bruke radio.on() som skrur på radiosambandet.

Til slutt må vi definere kva for ein kanal micro:bitane skal kommunisere på. Da må vi passe på å velge same kanal for alle micro:bitane som skal «snakke saman».


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


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

Når ein programmerer i Python, må ein gjere om tal til tekst (strenger) før det går an å sende dei over radio-sambandet. Det blir gjort med str()-kommandoen.


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

Denne kommandoen tek imot teksten som blir sendt og legg han i ein variabel kalla x.

Musikk
- med Python Mu

En micro:bit kan enkelt koplast til ein summar (også kalt buzzer) eller høgtalar for å spele av lyd. Sidan ein summar berre kan spele av ein enkelt tone, vil det bli mykje finare lyd frå ein høgtalar, da han kan spele dei overtonane vi er vane med å høyre som ein del av dei reine tonane i musikk.

For å kunne bruke en høgtalar eller ein summar i Python Mu, må vi importere eit ferdig bibliotek som inneheld metodar for å lage og kontrollere lyd.

```
from music import *
```

Når vi skal spele av ein tone i Python Mu, må vi oppgi tre ting som tekst (strenger). Tonen sitt namn, kva for ein oktav han høyrer til og kor lang tid han skal spelast. Eit piano dekker om lag sju oktavar, MakeCode for micro:bit dekker tre oktavar og Python Mu dekker ni oktavar. Oktav 0 er den lågaste oktaven, som gir dei mørkaste tonane. Oktav nummer fire er den som inneheld kammertonen A med frekvens 440 Hz. I MakeCode tilsvarer låg oktav (liten) oktav nummer tre, midtre oktav (einstroken) oktav nummer fire, og høg oktav (tostroken) oktav nummer fem i Python Mu. Lengda på tonen vert oppgitt som eit tal som svarar til talet på slag multiplisert med 4, der lengda på kvart slag er definert av Mu. Funksjonen under speler av fire tonar.

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

Namn på tonen

Kva oktav tonen tilhøyrer

Kor mange slag tonen skal spelast

Dersom vi ønsker å skrive litt mindre, kan play()-funksjonen hugse kva for ein oktav vi bruker og kor mange slag tonen varer. Dette er særs nyttig å bruke når vi skriv heile melodiar, da blir det ikkje fullt så mykje å skrive. Som regel vil ein melodi i hovudsak vere del av berre éin oktav. Når vi nyttar Mus minne, ser kommandoen over slik ut:

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

Dersom du vil avgjere sjølv akkurat kor lenge kvar tone skal spelast, kan du bruke ein funksjon som blir kalla pitch(). Da må du oppgi frekvensen til tonen, og ikkje namnet på han. Nokre frekvensar er oppgitt i tabellen i opplegg 11. Speletida blir oppgitt i millisekund, og med det kan du avgjere tempoet (bpm) med stor presisjon.

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

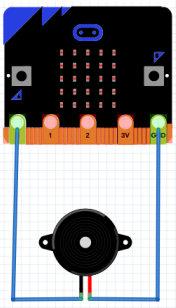
Frekvensen

Varighet i millisekunder

Kva for ein utgang høgtalaren (summeren) er kopa til. Denne treng du ikkje skrive noko på dersom du nytter pin0, da den ligg inne som standardverdi.

Det finnest og ei rekke ferdige melodiar. Du kan finne dei på denne sida: https://microbit-micropython.readthedocs.io/en/v1.0.1/tutorials/music.html

Det har ikkje noko å seie kva utgang på høgtalaren eller summaren du koplar til kva for ein utgang på micro:biten.



Oppgåve

Lag eit program som speler av «Lisa gikk til skolen».

Lag ein eigen melodi som du speler av på micro:biten.

Logge til PC/MAC med Python Mu

Noko det kan være veldig fint å bruke micro:bit til, er som ein dataloggar. Det vil seie at vi nyttar han til å samle inn data, slik vi har gjort mykje av, og så får vi datamaskinen til å skrive opp målingane på skjermen og lage ein graf av dei medan vi måler.

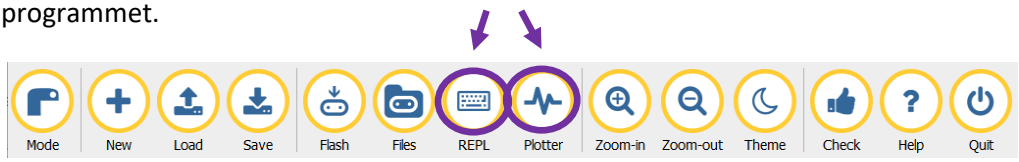
For å vise noko på skjermen bruker vi print()-kommandoen. Det som står inni parentesen er det som kommandoen skriv til skjermen, altså viser på skjermen. Dersom vi berre er interesserte i å få ut ei liste med målingar etter kvart som dei blir målte, kan vi berre bruke ein kodesnutt som ser slik ut:

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

Måler temperaturen og legg verdien i variabel x.

Skriv ut verdien av variabel x til skjermen.

Etter at vi har ført heile programmet over til micro:biten, må vi trykke på to knappar for å få fram tabellen med målingane. Trykk først på «Plotter», deretter på «REPL». Det er viktig å gjere dette i denne rekkefølga, sjølv om vi ikkje skal vise nokon graf (plottaren viser grafen), elles krasjar programmet.

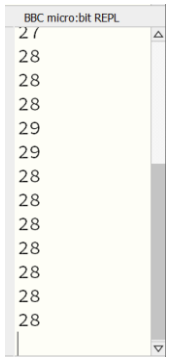


Ofte vil vi heller sjå grafen for målingane, og da må vi endre litt på koden vår. I Python Mu vil vi få vist grafar, dersom vi har rett format på det vi skriv ut til skjermen. Dette formatet blir vist under.

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

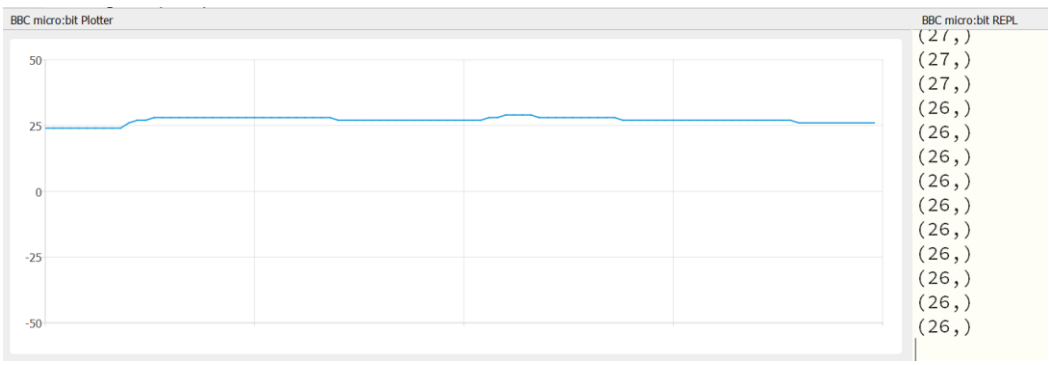
Diskuter

Kva er skilnaden på dei to kodesnuttane på denne sida?



Slik ser da lista med målingane ut.

Når det nye programmet er ført over, og du åpnar plottar, deretter REPL, vil vi sjå både grafen og ei liste over verdiane. Denne lista er litt meir tungvint å lese ut frå, men så lenge ein klarer å sjå bort frå ekstra parentesar og komma, står verdiane frå temperaturmålingane som på figuren under.



Oppgåve

Lag heile programmet som måler temperaturen og plottar han som ein graf på skjermen.

Kva skjer om du skriv inn ein annan verdi i staden for eit tomt felt i print()-kommandoen?

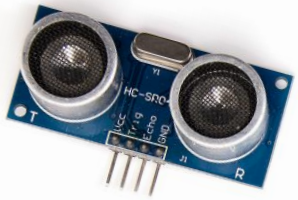
Diskuter

Å logge data kan vi gjere med alle dei innebyggede sensorane til micro:bit, men og med alle sensorar vi kopler på sjølv.

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

Avstandsmålar (sonar)

- med Python Mu

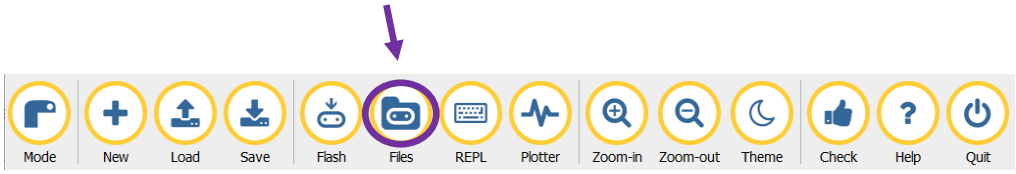


Ein sonar sender ut eit lydsignal, og måler kor lang tid det tek før signalet blir reflektert attende til sonaren. Dersom ein veit kva lydfarten er, kan vi enkelt rekne ut avstanden til det som reflekterer lydsignalet. Sonaren veit kva lydfarten er, og vi kan dermed bruke han til å måle avstandar med. For å kunne bruke ein sonar i Python Mu, må vi importere eit bibliotek på sjølve micro:biten. Dette biblioteket er eigentleg eit program som gjer det enklare for oss å programmere servoen, utan å måtte tenke på alle detaljane i målingane.

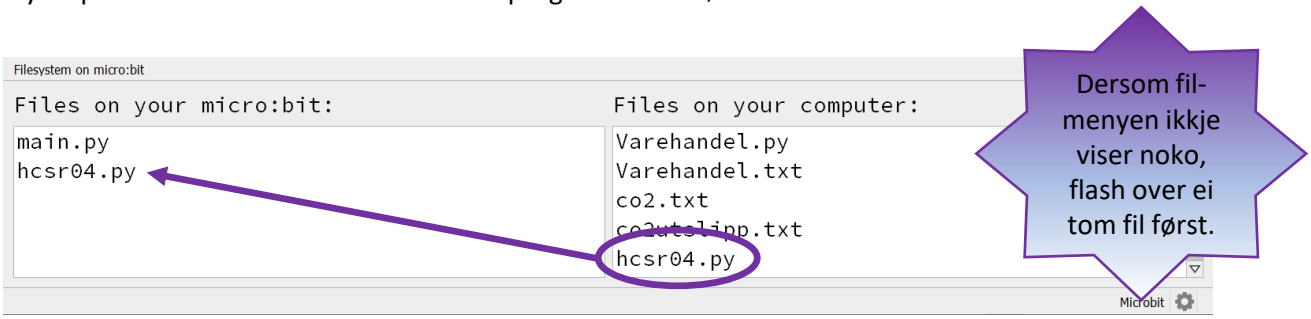
Det første vi må gjere er å få tak i biblioteket til sonaren, og det kan finnast på www.kunnskapsfilm.no\.....

Kopier koden som står på sida i ei ny fane (fil) i Python Mu og lagre denne som «hcsr04.py» med å trykke på «Save» og lagre fila i katalogen som dukkar direkte opp.

Så må denne fila leggast over på sjølve micro:biten, og det kan gjerast med å kople micro:biten til datamaskinen, og trykke på «Files».



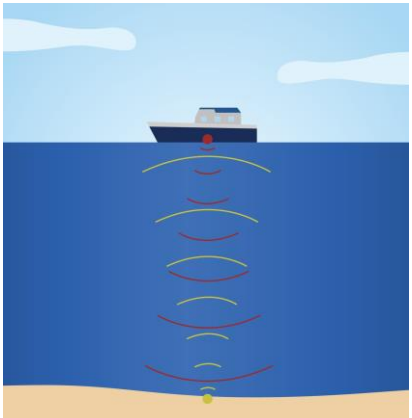
Da kjem denne menyen opp i Mu, og du må leite i feltet på høgre side til du finn fila som du nettopp lagra: «hcsr04.py». Når du finn henne, dreg du henne over i det venstre feltet. Da vil lyset på baksida av micro:biten blenke til programmet er ført over.



Da står det att å lage sjølve programmet som bruker sonaren til å måle avstandar med. Under er det ei puslespel-oppgåve der kodelinjene må ryddast i rett 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

Ein servo mottek eit signal frå micro:biten slik at han rører seg slik vi ønsker. Det finnest to typar servoar, 180° og kontinuerlege (360°), og de blir styrte på litt ulike måtar. 180°-servoen kan røre seg frå ein -90°-vinkel til ein 90°-vinkel, altså 180° til saman. For ein kontinuerleg servo gir vi opp kor raskt han skal snurre rundt i prosent. Sjølve farten kan variere litt med kor mykje straum som er att i batteria, for servoar trekker ganske mykje straum. Så dersom du har ein servo på 100 % fart i byrjinga av skuletimen, vil han snurre ein del saktere på slutten av timen.

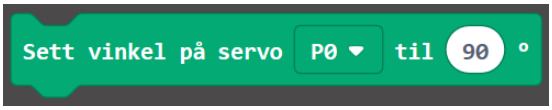
For å kunne bruke ein servo i Python Mu, må vi importere eit bibliotek på sjølve micro:biten. Dette biblioteket er eigentleg eit program som gjer det enklare for oss å programmere servoen, utan å måtte tenke på alle detaljane i signala som styrer han, på same måten som for sonaren.

Det første vi må gjere er å få tak i biblioteket til servoen, og det kan finnast på www.kunnskapsfilm.no\.....

Kopier koden som står på sida i ei ny fane (fil) i Python Mu og lagre fila som «servo.py» med å trykke på «Save» og lagre henne i katalogen som dukker direkte opp.

Så må denne fila leggast over på sjølve micro:biten, og det kan gjerast slik som for sonaren på førre sida.

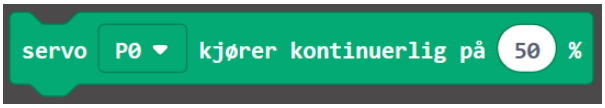
180°-servo




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

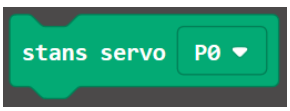
Med å bruke denne kommandoen, kan vi stille inn kva vinkel servoen skal røre seg til. Kommandoen tek inn kva for ein utgang vi har kopla servoen til, og kva vinkel servoen skal røre seg til. Vinkelen kan variere mellom -90° og 90°.

Kontinuerlig servo (360°)




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

Med å bruke denne kommandoen, kan vi stille inn kva for ein fart servoen skal snurre med. Denne kommandoen tek og inn kva utgang vi har kopla servoen til, i tillegg til farten i prosent. Farten kan variere mellom -100 % og 100 %, der forteiknet avgjer i kva retning servoen skal snurre.




```
servostopp(pin0)
```

Denne kommandoen får ein kontinuerlig servo til å slutte å snurre rundt. Det går og 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 modellar. Hadde det ikkje vore kult om vi kunne fått micro:biten til å lage slike filer av målingane sine? Da kan vi bruke micro:biten til å samle inn data over ein viss periode, utan å vere kopla til ein datamaskin. Datasettet blir lagra i micro:biten som ei .txt-fil, og vi kan legge fila over i datamaskinen, før vi plottar resultata eller lager ein matematisk modell med Python.

Det aller første vi må gjere, er å opprette ei tekstfil, det vil seie ei fil som sluttar på .txt, og det er enklast å gjere i Mu. Lagre fila i same mappe som resten av Python Mu-filene dine.

Deretter må den fila du har laga, kopierast over på micro:biten. Dette kan du gjere på same måten som for sonaren.

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

Åpner fila som heiter filnavn.txt og tildeler henne til eit objekt som vi kallar fil. Sidan vi har 'w' (write) i kommandoen, blir det gjort klart til å skrive inn i fila.

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

Tilsvarande som linja over, men sidan vi har 'r' (read) i kommandoen, gjerast det klart til å lese fila. Det er valfritt å ta med 'r'.

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

Skriv variabelen y til fila vår. +'n' gjer at det blir eit linjeskift etter kvart tal, og det må vi ha for at python skal kunne lese filene. Må stå med innrykk.

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

Gjer om variabelen x til ein streng (tekstverdi). Må stå med innrykk.

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

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

```
print(z)
```

Skriv variabelen z til skjermen på PC. Må IKKJE stå med innrykk.

Da står det att å lage sjølve programmet. Under er eit døme der micro:biten måler temperaturen og legg målinga i eim.txt-fil ved namn datasett.txt. Det kan vere greit å kunne sjekke korleis datasett.txt-fila ser ut utan å overføre til datamaskinen og åpne henne, difor er dette og med i puslespel-oppgåva.

Puslespel-oppgåve

```
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 *
```

Ekstraoppgåve

Lag eit program som bruker ein sensor du koplar på micro:biten for å samle inn data som du lagrar i ei fil på micro:biten. Kva må du endre på?

Kva må du endre i programmet for å lage ei fil med to kolonner der den første kolonna er kor lang tid det har gått før målinga er gjort?