
# Programmering

## Micro:bit grunnopplæring


*Gunnar Knutsen 2023*

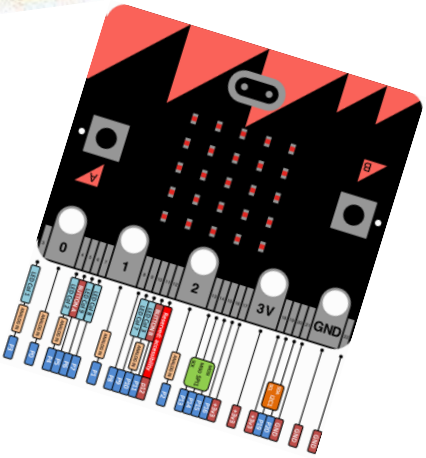
Innholdsfortegnelse

# Innledning

Denne delen handler i hovedsak om å bli kjent med programmering gjennom prosessorenheten Micro:bit. Micro:bit kan bruke blokkprogrammering som er basert på Java, men også andre programmeringsspråk. Til alle eksemplene er det lagt vekt på at vi arbeider parallellt med innlæringselementene i Modul 1, dvs. Algoritmer, Boolsk algebra, flytskjema og ser på forskjellen mellom blokk og Phyton programmering.

Du vil også lære litt om datamaskin, grunnleggende elektronikk, sensorer, måleteknikk i tillegg til programmering. Du vil også lære hvordan man kan kombinere Software og maskinvare, hvordan vi kan kommunisere med omverdenen ved hjelp av dataprogrammer og hvordan vi kan lage ulike øvelser.


MicroBit er en liten" datamaskin" som kan registrere og styre fysiske omgivelser og komponenter, ved at man kobler den til ulike sensorer og utgangsenheter. Dette gjør MicroBit til en egnet enhet for å lære programmering for både nybegynnere og opp til mer avansert bruk/nivå.




# Hva trenger du?

For å komme i gang med MicroBit trenger du følgende:

- PC (Windows, Mac, Linux).
- MicroBit Versjon en eller versjon to, eller en startpakke med alt man trenger for å komme i gang, her finnes mange muligheter å få kjøpt.
- Programvare (lastes ned fra nett).
- Diverse komponenter for å realisere funksjonene i programmeringen.

For nedlastning av MicroBit IDE plattform (Integrated Development Environment)

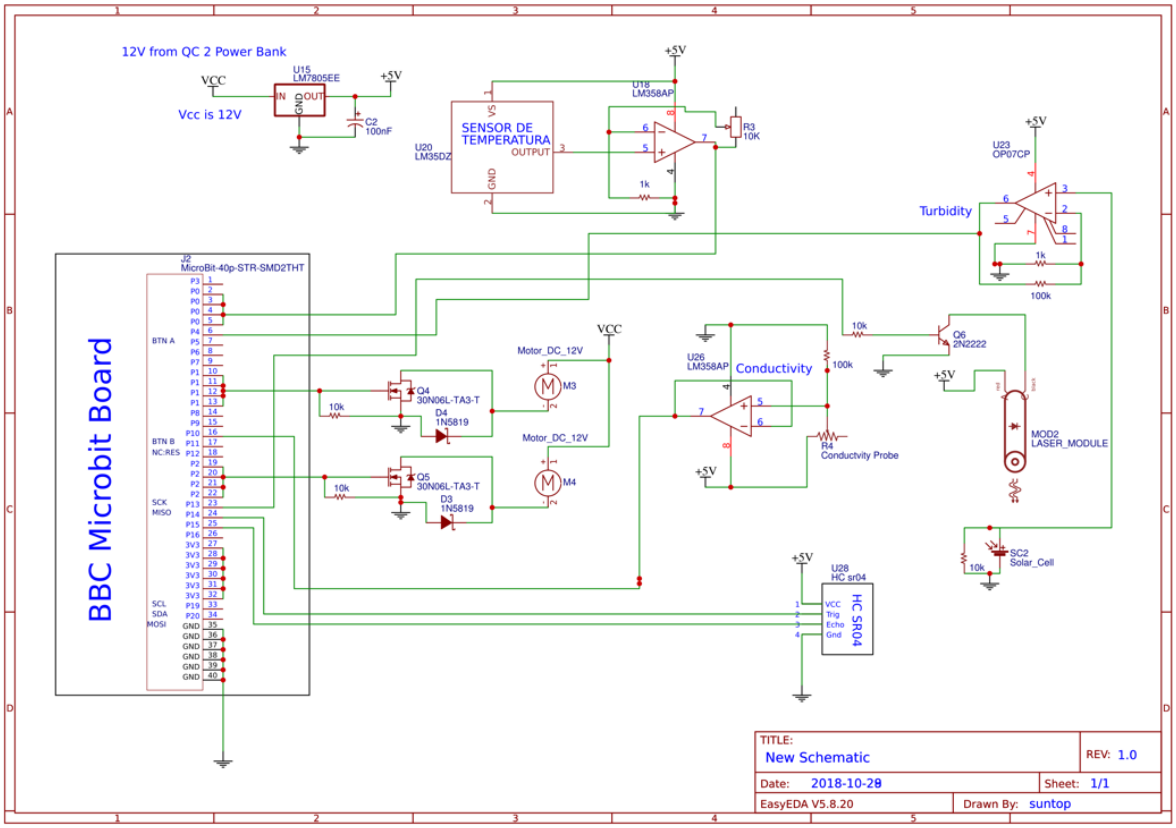
Oppstart og Introduksjon

MicroBit er en Open Source plattform som benyttes for å lære grunnleggende programmeringsteknologi, utvikle prototyper og grunnleggende elektronikk.

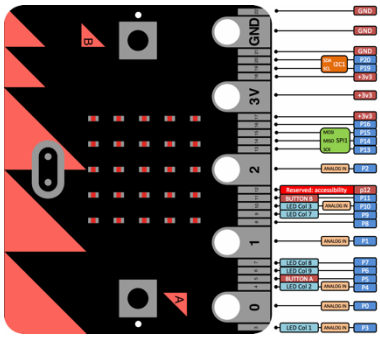
Plattformen består av en familie med små" datamaskiner", et programmeringsspråk med tilhørende utviklingsmiljø (IDE), og en rekke sensorer, aktuatorer og" Shields" som gjør at man kan bygge grensesnitt mot den fysiske verden.

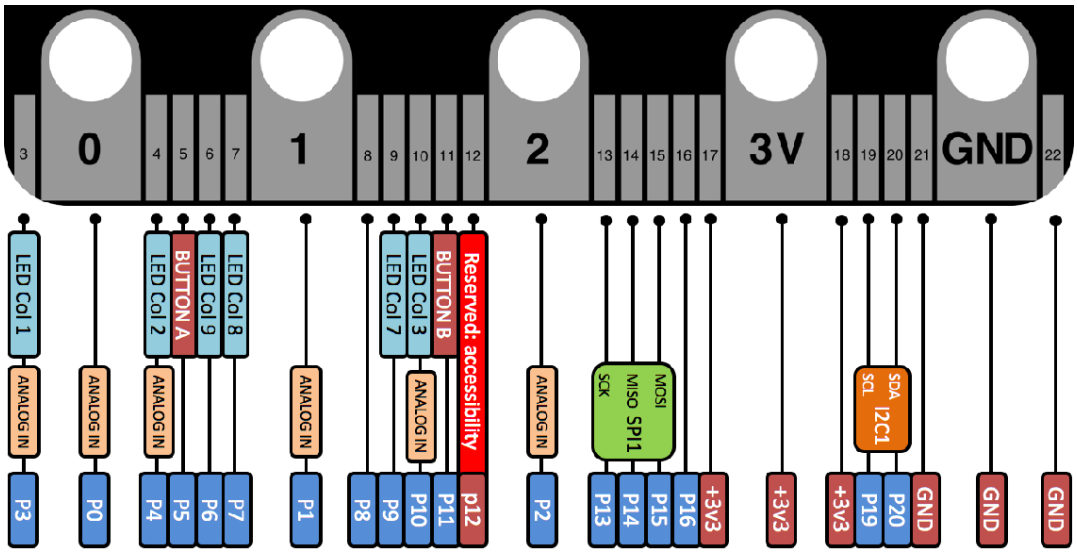
https://archive.microbit.org/no/ideas/

https://www.hackster.io



## Inn og utganger Micro: bit






## Tilkobling av Micro: bit til Pc

Tilkopling av Pc og nedlastning av programvare gjennomgås på linken oppstart og Introduksjon.

- Oppstart og Introduksjon

## Kort Om MicroBit

BBC MicroBit er en håndholdt, programmerbar mikrodatamaskin med en skjerm på 25 lysdioder. Den har innenbygget Bluetooth og ulike sensorer som kan programmeres. MicroBit kan kodes fra hvilken som helst nettleser i språkene Blokk, JavaScript, Python, Scratch og flere; ingen programvare kreves.

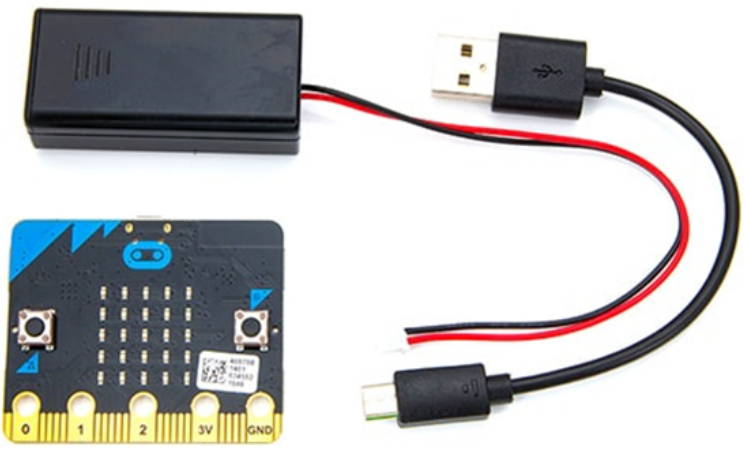
Enheten appelerer til en enkel og morsom inngang til programmering og prototype utvikling, det er bare å slå den på, programmer den til å gjøre noe morsomt – bruk det, tilpass det, utvikl det.
Det er en rekke muligheten for å koble til sensorer, skjermer og flere andre enheter, kan den brukes til alle slags kreasjoner, fra roboter til musikkinstrumenter – mulighetene er mange.

For mer informasjon: https://microbit.org/

**Mål**

Ved slutten av denne delen skal du kunne beskriv og bruke følgende programmeringskonsepter:

- Algoritmer
- Variabler
- Betingelser
- Gjenntakelser
- Koordinater
- Boolsk algebra
- Biter, byte og binære koder
- Radio

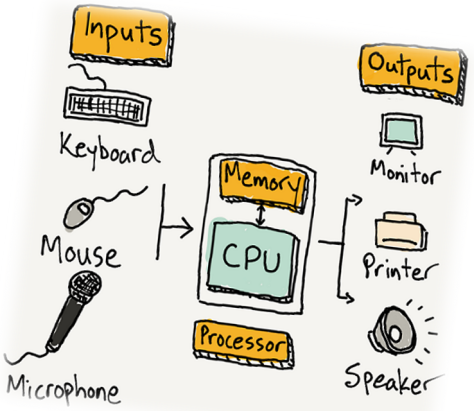



Merk: Hvis du vil ha mer informasjon om tekniske opplysninger og samsvarsforhold, kan du se: microbit.org/guide/hardware/.

Merk: En gratis MakeCode for MicroBit Windows 10-app er også tilgjengelig hvis skoleenhetene dine bruker Windows 10-operativsystemet. Den vil laste ned MakeCode-programmer direkte til Micro:bit uten å måtte dra og slippe programfilene dine til USB-stasjonen. Du finner valgfri programvare på aka.ms/microbitapp.

# Hvordan fungerer datamaskinen?

Hva er en datamaskin?




Det er normalt fire hovedkomponenter som utgjør en datamaskin:

1. **Prosessoren** - er vanligvis en brikke inne i datamaskinen, og det er ved hjelp av denne datamaskinen behandler og transformerer informasjon. Har noen hørt om begrepet "CPU"? CPU står for sentral prosesseringsenhet (Central Prossor Unit). Du kan tenke på prosessoren som hjernen til datamaskinen - jo raskere prosessoren er, jo raskere kan datamaskinen behandle data.

2. **Minnet** - slik husker datamaskinen ting. Det finnes to typer minne:

- RAM (random access memory) – som er datamaskinens kortsiktige minne, ting som er lagret her vil forsvinne når datamaskinen er slått av.
- Lagring (også kalt datamaskinens «harddisk") - dette er datamaskinens langsiktige minne, hvor den kan lagre informasjon selv når strømmen er slått av.

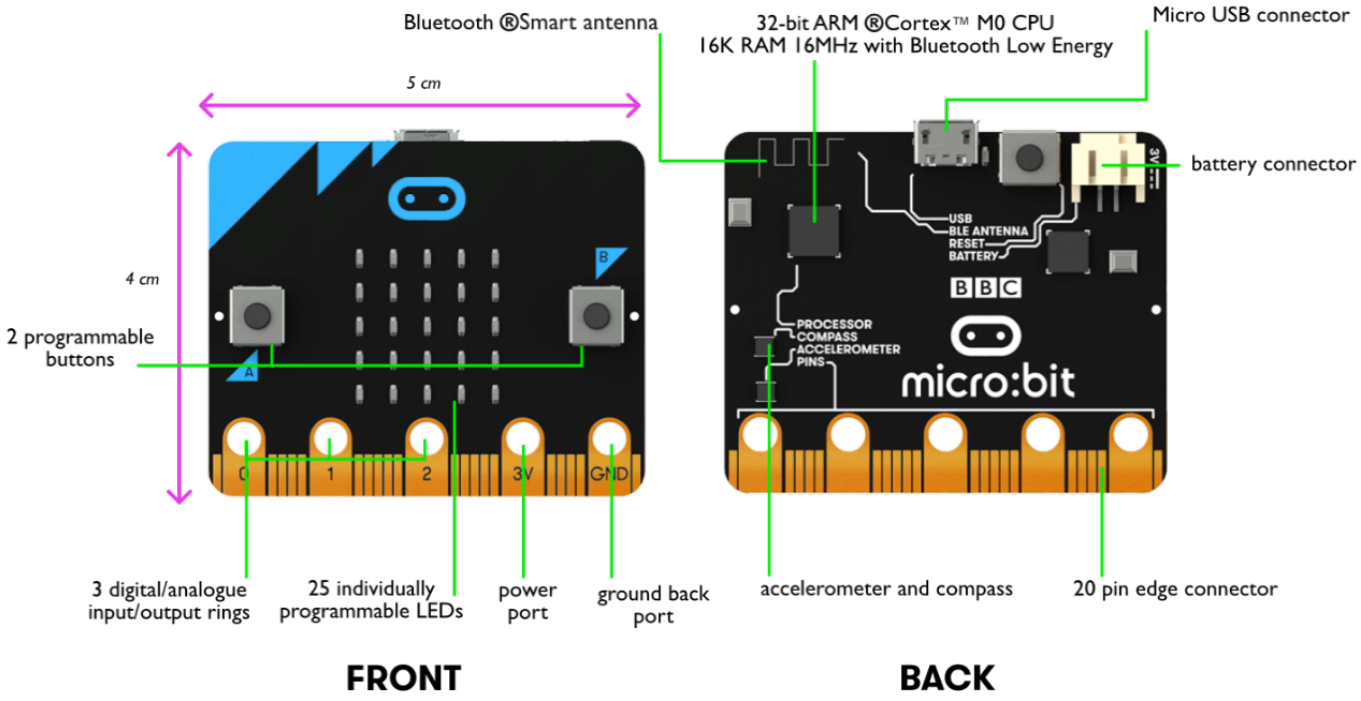
3. **Innganger** – slik henter datamaskin informasjon fra verden. Typiske innganger er tastatur, mus, berøringsskjermer, kamera, mikrofon, spill kontroller, scanner, etc.

4. **Utganger** – slik kommuniserer en datamaskin til omverden. Noen eksempler på kommunikasjon eller makinens utganger er skjerm,høretelefoner, høyttalere, printer og lignende

La oss nå se på micro:bit sine inn og utganger [microbit.org/guide/features/](microbit.org/guide/features/):

**Kunnskapssjekk!**

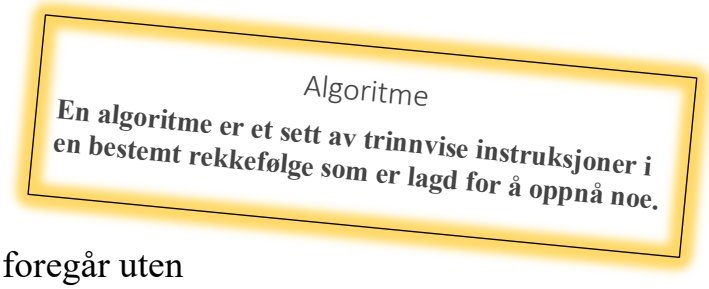
- *Hva er de fire hovedkomponentene som utgjør en datamaskin?*
- Hvor mange programmerbare knapper er på micro:bit?
- Hvor mange LED har Microbit?
- Har micro:bit innebygget lys føler?
- Hvor stort er RAM minnet til micro:bit?
- Hva er et accelometer?
- Har micro:bit innebygget Bluetooth?
- Har micro:bit innebygget temperatur føler?




# NB! Micro:Bit V2

# ALGORITME

## 1. Frakoplet aktivitet
## Algoritmer




En frakoblet aktivitet er en aktivitet som foregår uten datamaskinen. Poenget med denne øvelsen er å bevege seg, reagere og samhandler med medelevene mens du utfører utfordringen. Meningen er å utføre øvelser slik at man enklere forstår oppsettene i en algoritme når man kopler til en datamaskin.

**Algoritme test- input/output!**
2 og 2 elever sammen. Bruk kun heltall, Integer, i programmeringsspråket

Eksempel startsum = 100

Elev A: Angir et tall som er svaret på oppgaven, eksempelvis 50
Elev B: Angir da «dele 100 på 2» som svar

Skriv ned alle funksjonene fra begge i loggen, bruk maksimalt 2 ledd i hvert trinn.

**Start sum: for elven A 250**

| Elev A | Elev B | Nytt utgangspunkt |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

NB! Kun for å vise størrelsesforholdet, ikke pensum på dette trinnet!


## 2. Frakoplet aktivitet Algoritmer

Dette er en interaktiv øvelse for å introdusere betingede og hendelsesbehandlinger.

Les gjennom hele aktiviteten og utfør i betinget rekkefølge

A. **Prosess Gruppe 1 og gruppe 2**
B. Prosessen starter når lærer skriver START på tavlen.

C. Hvis det står START på tavlen, skal elev som sitter på plass A reise seg , ta med den blå white board fargen og gå opp til tavlen og skrive en A på tavlen, og sette seg på plassen merket C.

D. Hvis elev A har skrevet A på tavlen, skal elev på plass B hente White Board fargen grønn på plassen merket C, og skrive bokstaven L etter bokstav A på tavlen, og sette seg tilbake på sin plass igjen.

E. Når elev B har satt seg skal den som sitter på plass C reise seg og slå på lyset.

F. d1. Hvis lyset er på skal den som er på plass C sette seg igjen.
G. d2. Hvis den som er på plass C setter seg igjen, skal elev på plass D reise seg og gå opp til lysbryteren å slå av lyset og så bli stående ved lysbryter.

H. Hvis lyset er slått av skal den som sitter på plass E reise seg og hente en rød white board farge på plass C, levere den til eleven som står ved lysbryteren og sette seg på plass D.

I. Eleven som har den rød White Board fargen og står ved lysbryteren, går nå til tavlen og skriver en G etter L på tavlen og setter seg på plassen E.

J. Hvis det står ALG på tavlen i fargene blå, grønn, rød og den som sitter på plass E har P i første bokstav i fornavn, skal den som sitter på plass F gå til plass C å hente en blå white board farge, gå opp til tavlen og skrive en O etter G på tavlen.

K. g1. Hvis eleven ikke har P som første bokstav skal den som sitter på plass E reise seg opp og sette seg 3 ganger etter hverandre.

L. g2. Så skal den som sitter ved plass F gå til plass C å hente en blå white board farge, gå opp til tavlen og skrive en O og R etter ALG.

M. Hvis det står ALGOR på tavlen, skal den som sitter på plass G, reise seg og gå til plass C, ta med seg en rød white board farge og gå til tavlen og skrive ITME etter ALGOR, og så sette seg på plass G.

N. Gjenta prosessen med gruppe 2 og startbokstav J

## 1. Tilkoplet aktivitet for å gå gjennom oppsett og program

Micro:bit er en maskinvare, som trenger et program eller en kode for å fungere. Programvaren "forteller" maskinvaren hva den skal gjøre, og i hvilken rekkefølge du skal gjøre det, ved hjelp av algoritmer. Algoritmer er sett med instruksjoner.

I denne aktiviteten vil du lære hvordan du bruker micro:bit-knappene som inngangsenheter, og skriver kode som får noe til å skje på skjermen som er utgang(de 25 lysdiodene). Vi vil også lære om pseudokode, MakeCode-verktøyet, hendelsesbehandlinger og hvordan man kan kommentere kodingen.

### Pseudokode

Hva vil du at programmet skal gjøre? Det første trinnet i det å skriver et dataprogram, er å opprette en plan. En plan for hva du vil at programmet skal gjøre. Skriv en detaljert, trinnvis plan for programmet. Planen bør inneholde hvilken type informasjon programmet skal motta, hvordan disse inndataene skal behandles, hvilke utdata programmet oppretter og hvordan utdataene skal registreres eller presenteres.

Du trenger ikke å skrive fullstendige setninger, og inkluderer heller ikke faktisk kodene. Denne typen skriving er kjent som pseudokode. Pseudokode er som et **grovt utkast** til programmet ditt. Pseudokode er en blanding av naturlig språk og kode.

Eksempel på en pseudokode:

- Start med en tom skjerm
- Når brukeren trykker på knapp A, vis et lykkelig ansikt
- Når brukeren trykker på knapp B, vis et trist ansikt

## Microsoft MakeCode

Nå som du har en plan for programmet ditt, i form av pseudokode, kan du begynne å lage det virkelige programmet i Microsoft MakeCode. Husk at MakeCode verktøyet kalles en IDE (Integrated Development Environment) og er et program som inneholder det du som programmerer trenger for å lage, kompilere, kjøre, teste og til og med feilsøke et program.

Start et nytt prosjekt: https://makecode.microbit.org/#editor

Prosjekt bibliotek: http://elecfreaks.com/learn-en/

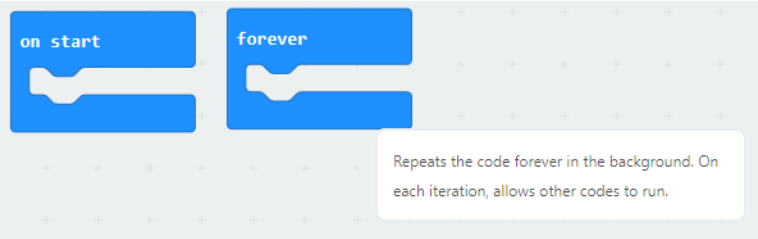
## Hendelsesbehandlinger

Når du starter et nytt prosjekt, vil det være to blå blokker, "ved start" og "for alltid" i arbeidsområdet. Disse to blokkene er *hendelsesbehandlinger fra Basisbiblioteket*.
I programmering er en *hendelse* en handling som gjøres av brukeren, for eksempel å trykke en tast eller klikke en museknapp. En *hendelsesbehandling* er en rutine som svarer på en hendelse. En programmerer kan skrive kode som forteller datamaskinen hva den skal gjøre når en hendelse inntreffer.

## Verktøytips

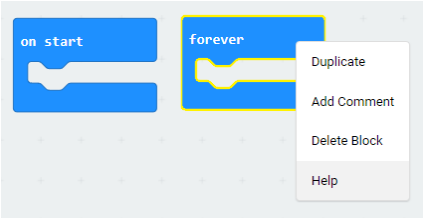
## Eksperimenter med følgende alternativer.

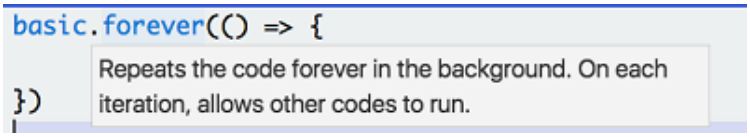
1. Blokker beskrivelser - Hold pekeren over en hvilken som helst blokk til et håndikon vises, og en liten tekstboks dukker opp og forteller deg hva den blokken gjør. Du kan prøve dette nå med blokkene "ved start" og "for alltid".




Hvis du holder pekeren over koden i JavaScript, har det samme effekt.

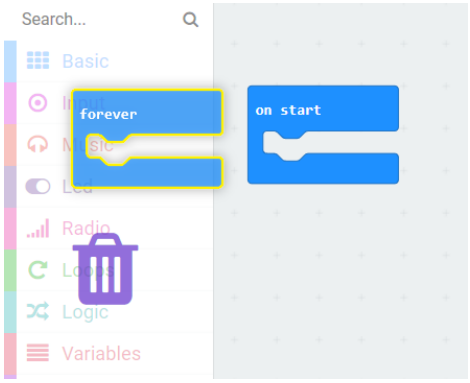
2. Hjelp – Du kan også høyreklikke en hvilken som helst blokk og velge Hjelp for å åpne referansedokumentasjonen.



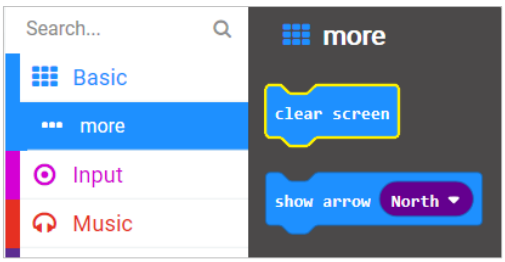

3. Slette blokker - Klikk på "for alltid" -blokken og dra den til venstre til Toolbox-området. Du skal se et søppelkasseikon vises. Slipp blokken, og den skal forsvinne. Du kan dra en hvilken som helst blokk tilbake til Verktøykasse-området for å slette den fra kodeområdet Workspace. Du kan også fjerne en blokk fra kodearbeidsområdet ved å:

- en mac). Hold pekeren over blokken, høyreklikk og velg Slett blokk, eller
- Velge blokken og deretter trykke på "slett" -tasten på tastaturet (eller kommando-X på

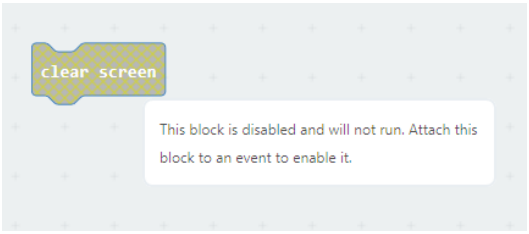



## *Fjern skjerm*

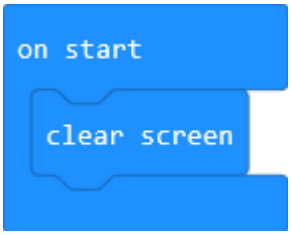
Når vi ser på pseudokoden vår, vil vi sørge for å starte et program med en klar skjerm. Vi kan gjøre dette ved å gå til Basic-menyen - mmalm og velge en "clear screen" -blokk. Dra «Tøm skjerm»-blokken til kodingsområdet. Legg merke til at blokken er nedtonet. Hvis du holder pekeren over "nedtonet"-blokken,vises enpopup-tekstbokssom forteller deg at siden denne blokken ikke er knyttet til en hendelsesbehandlingsblokk, vil den ikke kjøre.



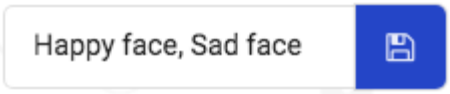
Gå videre og dra 'tøm skjermen' -blokken inn i 'ved start'-blokken. Nå er blokken ikke lenger nedtonet, noe som indikerer at den vil kjøre når hendelsen inntreffer, det vil si at programmet starter.




Vi har nå et arbeidsprogram som kjører på micro:bit simulator! Når du skriver programmet ditt, vil MakeCode automatisk kompilere og kjøre koden din på simulatoren. Programmet gjør ikke mye på dette tidspunktet, men før vi gjør det mer interessant, bør vi navngi programmet vårt og lagre det.




Nederst til venstre i programvinduet, til høyre for Last ned-knappen, er en tekstboks der du kan gi programmet et navn. Når du har gitt programmet et navn, trykker du på lagre-knappen for å lagre det.

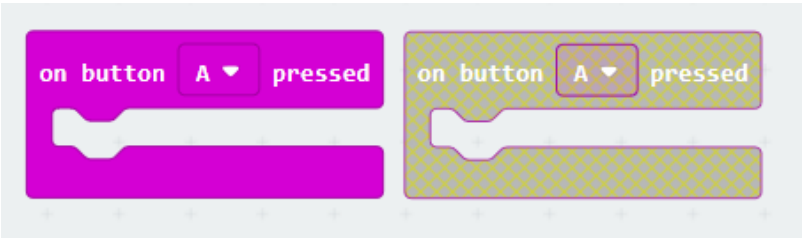



Viktig: Når du skriver et betydelig stykke kode eller bare noen få minutter, bør du lagre koden. Hvis du gir koden et meningsfylt navn, blir det raskere å finne den fra en liste over programmer, og du får vite hva programmet ditt gjør.

*Flere hendelsesbehandlinger*

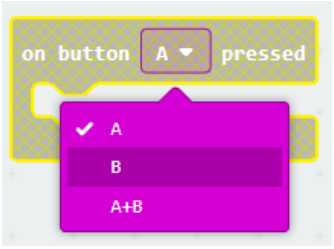
Nå for å gjøre programmet vårt litt mer interessant ved å legge til to hendelsesbehandlinger til.

Fra Inndata -menyen drar du to "on button A pressed"-blokker til kodearbeidsområdet. Legg merke til at den andre blokken er nedtonet. Dette er fordi de akkurat nå er den samme blokken, begge "lytter" for den samme hendelsen 'på knapp A trykket'.

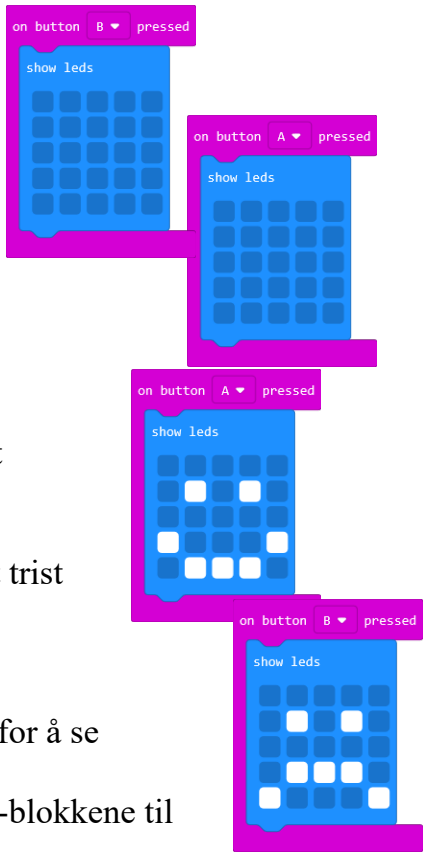



La den første blokken være i fred for øyeblikket, og bruk rullegardinmenyen i den andre blokken til å endre A til B. Nå vil denne blokken ikke lenger være nedtonet, da den nå lytter etter en annen hendelse, 'på knapp B trykket'.





## *Vis lysdioder*



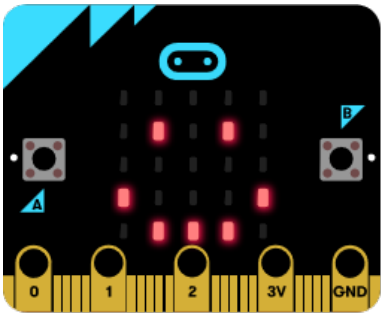

Nå kan vi bruke LED-lysene våre til å vise forskjellige bilder avhengig av hvilken knapp brukeren trykker på.
Fra Grunnleggende verktøykasse drar du to "show leds"-blokker til kode  Workspace.
Plasser en "show leds" -blokk i hendelsesbehandlingen "på knapp A trykket" og den andre "show leds" -blokken i hendelsesbehandlingen "på knapp B trykket".

Klikk på de enkelte små boksene i 'vis led'-blokken som er i hendelsesbehandlingen 'på knapp A trykket' for å lage bildet av et lykkelig ansikt.
Klikk på de enkelte små boksene i 'vis led'-blokken som er i hendelsesbehandlingen 'på knapp B trykket' for å lage bildet av et trist ansikt.

## *Test programmet ditt!*

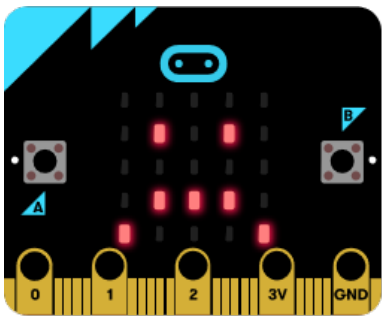
I simulatorensender du knappen A og deretter knappen B på nytt for å se utdataene som produseres av koden.
Føl deg fri til å leke med å slå lysdioder på eller av i "show leds" -blokkene til du får bildene du ønsker.

Husk å lagre koden.




## *Kommentere  koden*

Det er lurt å legge til kommentarer i koden. Kommentarer kan være nyttige på flere måter. Kommentarer kan hjelpe deg med å huske hva en bestemt kodeblokk gjør, og/eller hvorfor du valgte å programmere noe slik du gjorde.

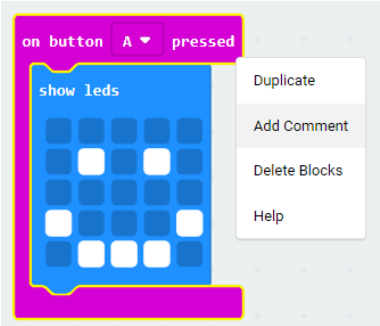



Kommentarer hjelper også andre med å lese koden din for å forstå de samme tingene.
Slik kommenterer du en kodeblokk:

- Høyreklikk på ikonet som vises før ordene i en blokk.
- En meny dukker opp. Velg Legg til kommentar.
- Dette vil føre til at et spørsmålstegnikon vises til venstre for det forrige ikonet.
- Klikk på spørsmålstegnet, og en liten gul boks vises der du kan skrive kommentaren din.

- Klikk på spørsmålstegnikonet igjen for å lukke kommentarfeltet når du er ferdig.

- Klikk på spørsmålstegnikonet når du vil se kommentaren din igjen eller redigere den.



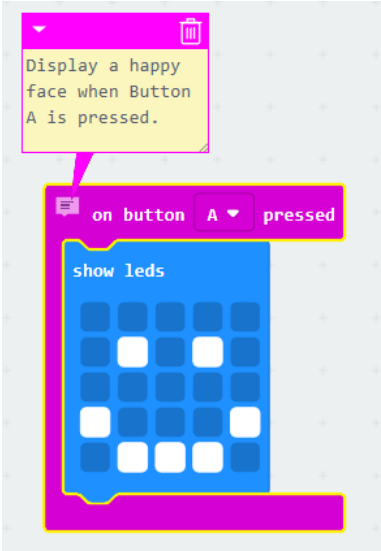

I JavaScript kan du legge til en kommentar ved å bruke to skråstreker og deretter skrive inn kommentaren. De to skråstrekene forteller JavaScript at følgende tekst (på samme linje) er en kommentar.

```
Vis et lykkelig ansikt når knapp A trykkes ned.
```

*Rydder opp!*

Rydd opp i kode-Workspace før du foretar en endelig lagring. Hva betyr dette?

- Det betyr at bare koden og blokkene du bruker i programmet, fremdeles finnes i arbeidsområdet.
- Fjern (slett) eventuelle andre blokker du har dratt inn i kodearbeidsområdet mens du eksperimenterte og bygde programmet.
-




*Lagre og last ned*

Nå som koden går helt fint i Simulator, er kommentert, og koding arbeidsområdet er "rent," lagre programmet, last det så ned til micro:bit en din og test!

Fullfør program

Løsningskobling:
makecode.microbit.org/_WPicbA7gR7xc

# Boolske elementer

Enten du oppretter formler i boolsk algebra eller bruker dem i programmene, danner du både enkle og komplekse logiske uttrykk som bruker grunnleggende operasjoner til å kombinere de logiske betingelsene.

**Notasjon**

Boolske (logiske) formler uttrykkes på en måte som ligner på matematiske ligninger. Variabler i boolske uttrykk har imidlertid bare **to mulige verdier, sanne eller usann**. For en formel som bruker et logisk uttrykk, vil de tilsvarende sidene av likhetstegnet , = , bare være sanne eller usanne også.

- $1010_2$
- $10_{10}$
- $F4_{16}$

Følgende liste viser de grunnleggende notasjonselementene for boolske uttrykk.

~A: den inverse (**NOT**) av A, når A er sann, ~ A er usann
A + B: verdien av A **OR** B
A · B : Verdien av A **AND** B
A⊕ B: verdien av den eksklusive OR (**XOR**) til A med B
Q: tilsvarende resultatverdi (OUTPUT) for et logisk uttrykk

En resultatverdi, Q , fra et logisk uttrykk i vises på denne måten:

Q =A+B
En formel som skal vise logisk like uttrykk (der begge sider har samme resulterende verdi) kan se slik ut: ~(A+B) = ~A·~B

Praktisk oppgave

- Kople opp de ulike logiske operatorene i Micro: Bit IDE og test dem.

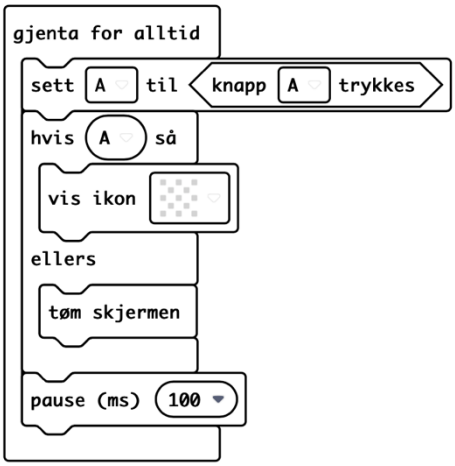
## Logiske operatorer

Alle boolske uttrykk er et resultat av en kombinasjon av betingelser og operatorer. Disse operatorene slår sammen individuelle forhold og evaluerer til én enkelt sann eller usann betingelse. De logiske operatorene vises her i både boolsk algebra, kode og sannhetstabell.

### identitet

Identitet betyr at en resultatverdi er den samme som selve betingelsen.

- **Q =A**
- **let A = false**
- **let Q = A**




### Eksempel - Blink LED-lamper på, trykket

| A | A |
|---|---|
| F | F |
| T | T |

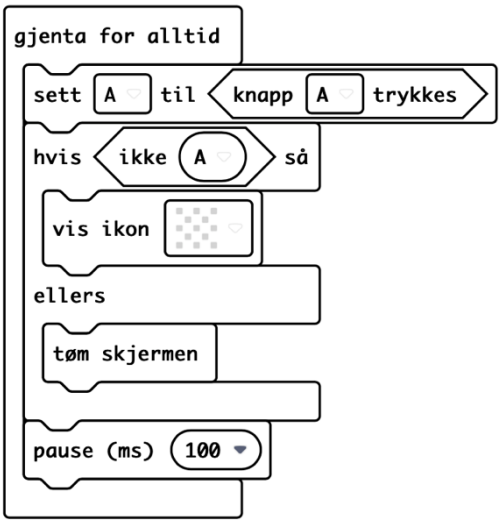
## IKKE (Negasjon)

NOT-operatoren kalles negasjon eller omvendt. Det tar en enkelt logisk verdi og gjør at den har motsatt verdi, sann går til usann og usann går til sann .

Q = ~A

let A = false

let Q = ! (A)




**Eksempel - Blink LED-lamper på, ikke trykket**

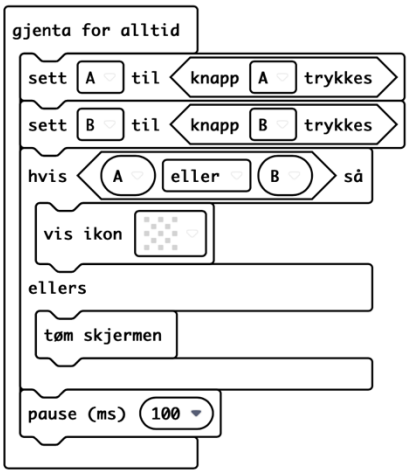
| A | ~A |
|---|---|
| F | T |
| T | F |

## OR (disjunksjon)

Opperatoren OR resulterer i sann når én eller flere betingelser er oppfylt

Q=A+B

let A = false
let B = false
let Q = A || B



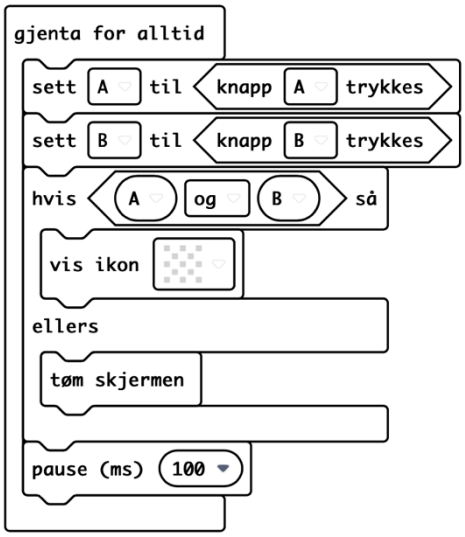

**Eksempel - Blink på et hvilket som helst trykk**

| A | B | A + B |
|---|---|---|
| F | F | F |
| T | F | T |
| F | T | T |
| T | T | T |

## OG (konjunksjon)

AND-operatoren krever at alle betingelsene er sanne  for at resultatet skal være sant

- Q = A·B
- let A = false
- let B = false
- let Q = A && B




### Eksempel - Blink bare ved dobbelttrykk

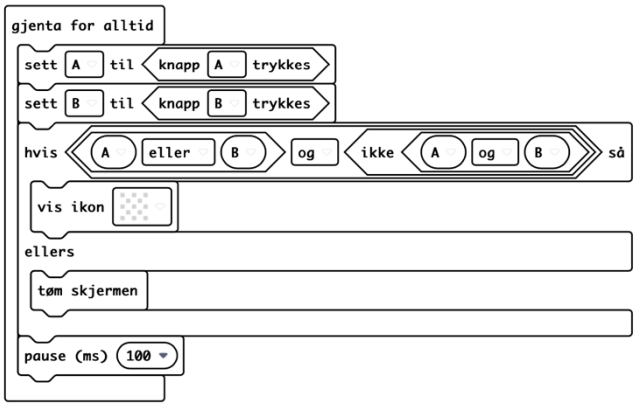
| A | B | A · B |
|---|---|---|
| F | F | F |
| T | F | F |
| F | T | F |
| T | T | T |

## XOR (eksklusiv ELLER)

Eksklusiv OR (XOR) betyr at bare én eller den andre betingelsen er sann. Begge forholdene kan ikke være sanne samtidig. XOR er vanlig i boolsk algebra, men den har ingen operator i JavaScript. Driften kan gjøres ved å kombinere noen få enkle uttrykk.

Q = A⊕B

let A = false
let B = false
let Q = (A || B) && !(A && B)




### Eksempel - Blink på ett trykk eller det andre

| A | B | A⊕B |
|---|---|---|
| F | F | F |
| T | F | T |
| F | T | T |
| T | T | F |

# Teknologien rundt MicroBit

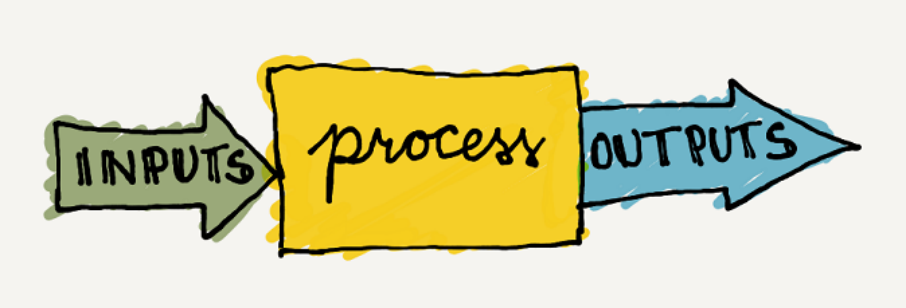
Micro: bit er basert på en liten ARM mikroprosessor som du kan programmere til å gjøre ulike oppgaver. Mikroprosessoren er montert opp på et brett med flere mulige innganger og utganger.

Programmeringsplattformen er tilgjengelig via nettlesere, nettbrett og mobil. Den er oversiktlig og gir mange muligheter til individuell tilpasning sammen med instruksjonene og oppgaver som ligger i programmene, samt innstillinger. Det finnes også et bredt utvalg av Biblioteker for ulike oppgaver og øvelser.

For å lære denne type teknologi starter man enkelt, men det er samtidig viktig å kjenne til forskjellen mellom Analog teknologi og Digital teknologi.
Grunnen til dette er at analog teknologi benyttes ofte i sammen med den digitale verden. Spesielt nå den digitale verden skal styre noe som krever annet en bare signaler med lav spenning og lite strøm.

Å starte enkelt kan være å skrive «Hallo verden» og se at noen lysdioder aktiveres, eller sende ut et signal og se at det tenner en liten LED på et koplingsbrett.

Uavhengig av hvordan vi starter er det viktig å ha en plan med undervisningen, og gjøre den «relevant» i forhold til faget, slik at man kan se nytteverdien og kunne utvikle denne videre i en eller flere retninger.



## Logiske funksjoner

### Variabler

- som lagringssted for data fra eksterne kilder som skal brukes i programmet, for eksempel sensor-data
- hvis programmet har kode som skal gjentas (løkker) et bestemt antall ganger
- for å slippe å manuelt skrive inn faste verdier som skal brukes mange ganger i løpet av koden
- hvis resultat fra utregninger gjort tidlig i koden skal brukes til utregninger senere i programmet
- for å lagre resultater fra koden som skal ut av programmet, for eksempel hvis de skal vises på skjermen til brukeren

### Konstant

- En konstant kan være en verdi som er aldri vil bli forandret, som for eksempel verdien av pi (3,141592). Det kan også være verdier som ikke vil bli forandret mens programmet kjører (men som kanskje skal forandres mellom versjoner av programmet).

For at datamaskinen skal vite når ulike operasjoner skal utføres, må du gi den et sett med betingelser. Med en if-setning kan vi sjekke om en betingelse er oppfylt før koden kjøres. Følgende kode sjekker om alderen til en person er mer enn eller er 18 år

JAVASCRIPT

```
if (alder >= 18) {

    // Kode kjøres dersom alder er større enn eller lik 18.

}
```

Hvis / if

Eksempel i Excel: Hvis en celle inneholder tall større en 3, skriv riktig, hvis ikke, sett feil.

=HVIS(A3>3;"riktig";"Feil")

| | | |
|---|---|---|
| 1 | | |
| | | |
| | | |
| | | |
| | | Feil |

| | | | |
|---|---|---|---|
| 4 | | | |
| | | | |
| | | | |
| | | | |
| | | | riktig |

Med en if-setning kan du sjekke flere logiske uttrykk etter hverandre.
Da bruker du **else if** ("ellers hvis" på norsk).

JAVASCRIPT

```javascript
if (alder >= 18) {

    // Kode kjøres dersom alder er større enn eller lik 18.

} else if (alder >= 15 {

    // Kode kjøres dersom alder er større enn eller lik 15.

} else {

    // Kode kjøres dersom betingelsen er usann.

}
```

Eksempel i Excel: Hvis en celle inneholder tallet 1, regn sammen A5 og B5 cellene, hvis ikke skriv tallet i Celle A3
=HVIS(A3=1;A5+B5;A3)
=HVISFEIL(A3=1;"usann")

| | | | |
|---|---|---|---|
| 1 | | | |
| | | | |
| 5 | 5 | | |
| | | | |
| | | | 10 |

| | | | |
|---|---|---|---|
| 2 | | | |
| | | | |
| 5 | 5 | | |
| | | | |
| | | | USANN |

## Og/and

JAVASCRIPT

```javascript
if (alder >= 18 && legitimasjon == true) {

    // Kode som kjøres dersom betingelsene er sanne

}
```

Og- funksjon brukes for å sette flere betingelser som må oppnås

## Eller/or

Eller benyttes for å gi valg til at den eller den

JAVASCRIPT

```javascript
if (alder >= 18 || hoyde >= 180) {

    // Kode som kjøres dersom betingelsene er sanne

}
```

## For-løkker

For-løkker bruker du dersom du ønsker at noe skal gjenta seg selv et gitt antall ganger. Nedenfor ser du et eksempel på hvordan en slik løkke kan se ut:

JAVASCRIPT

```javascript
for (let i = 0;i<10;i++) {
    // Kode som kjøres ti ganger
}
```

**While- løkker,** når du ikke vet hvor mange ganger løkken skal gå i forkant. Den kan også egne seg dersom betingelsen ikke baserer seg på tall, eller når du ikke vet hvor mange ganger loopen skal kjøre.

JAVASCRIPT

```javascript
let teller = 0;

while (teller <10) {
// Kode som kjører ti ganger
teller++;
}
```

## Funksjoner

JAVASCRIPT

```javascript
function LeggSammen(tall1, tall2) {
    return tall1+tall2;
}
```

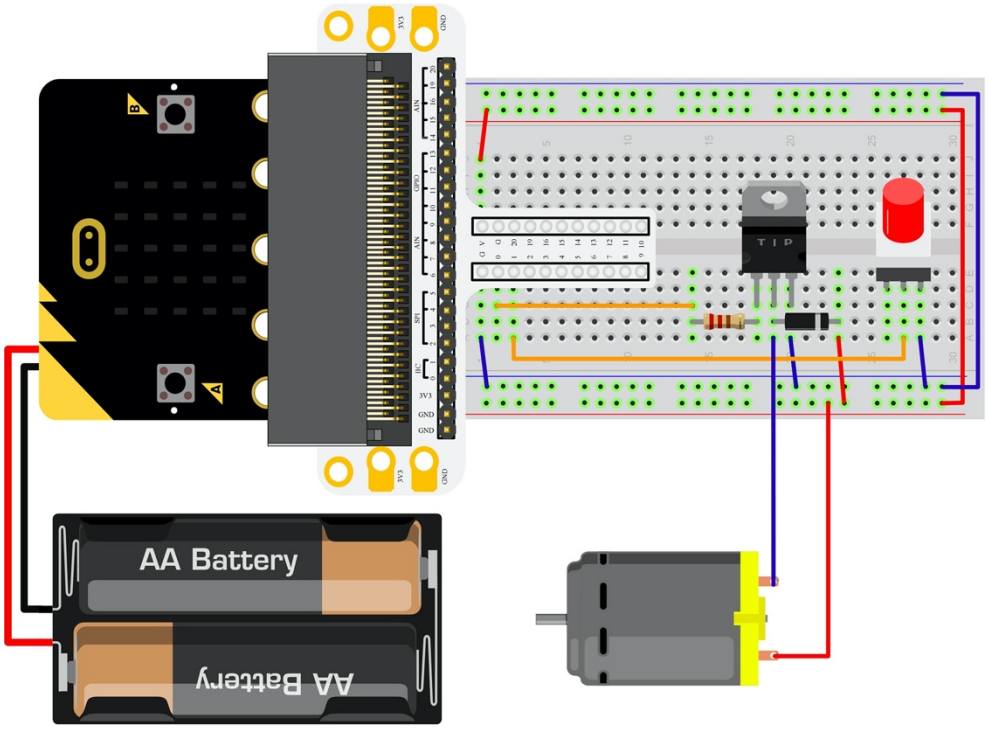
- Legg sammen tall1 og tall 2 osv.
- Return gir summen av tall 1 + tall 2

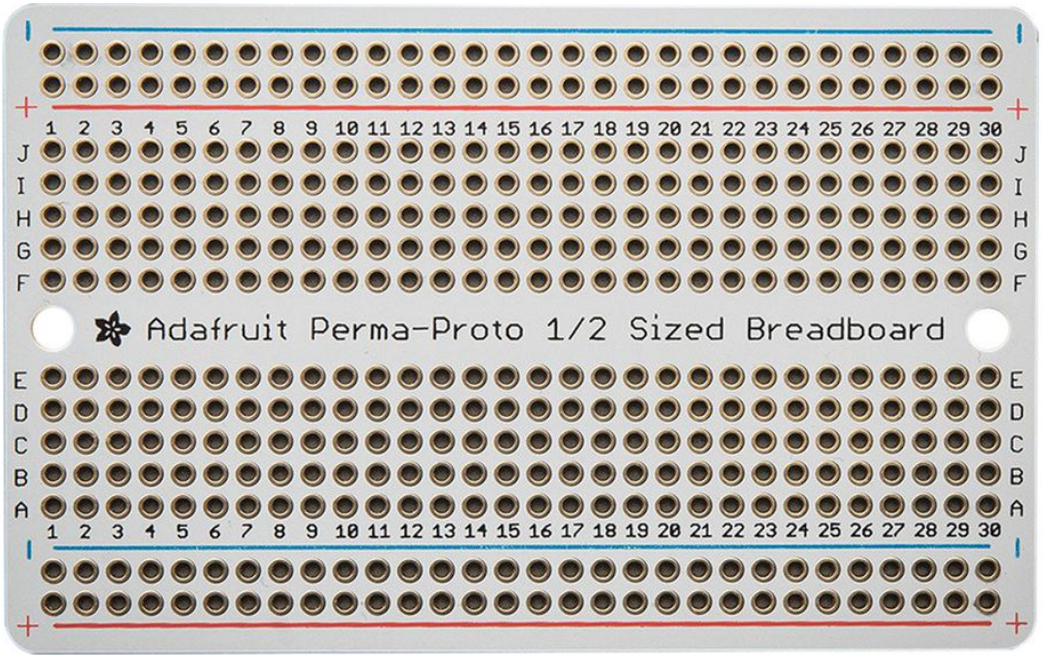
## Kommentering av koden

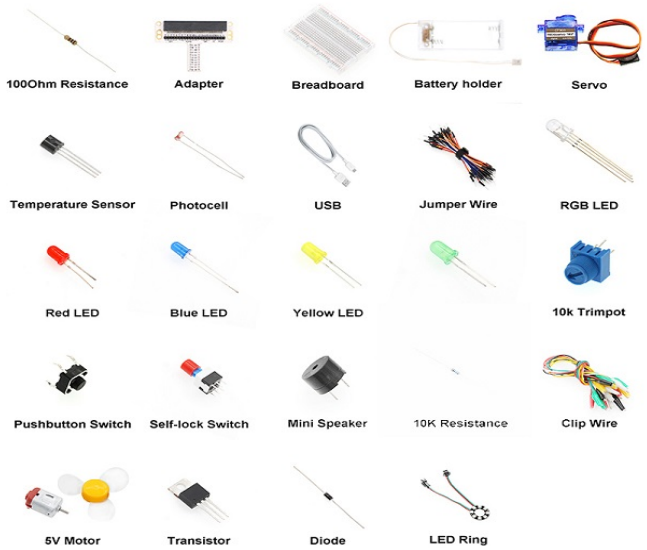
JAVASCRIPT

```javascript
// Dette er et eksempel på en kommentar.
```

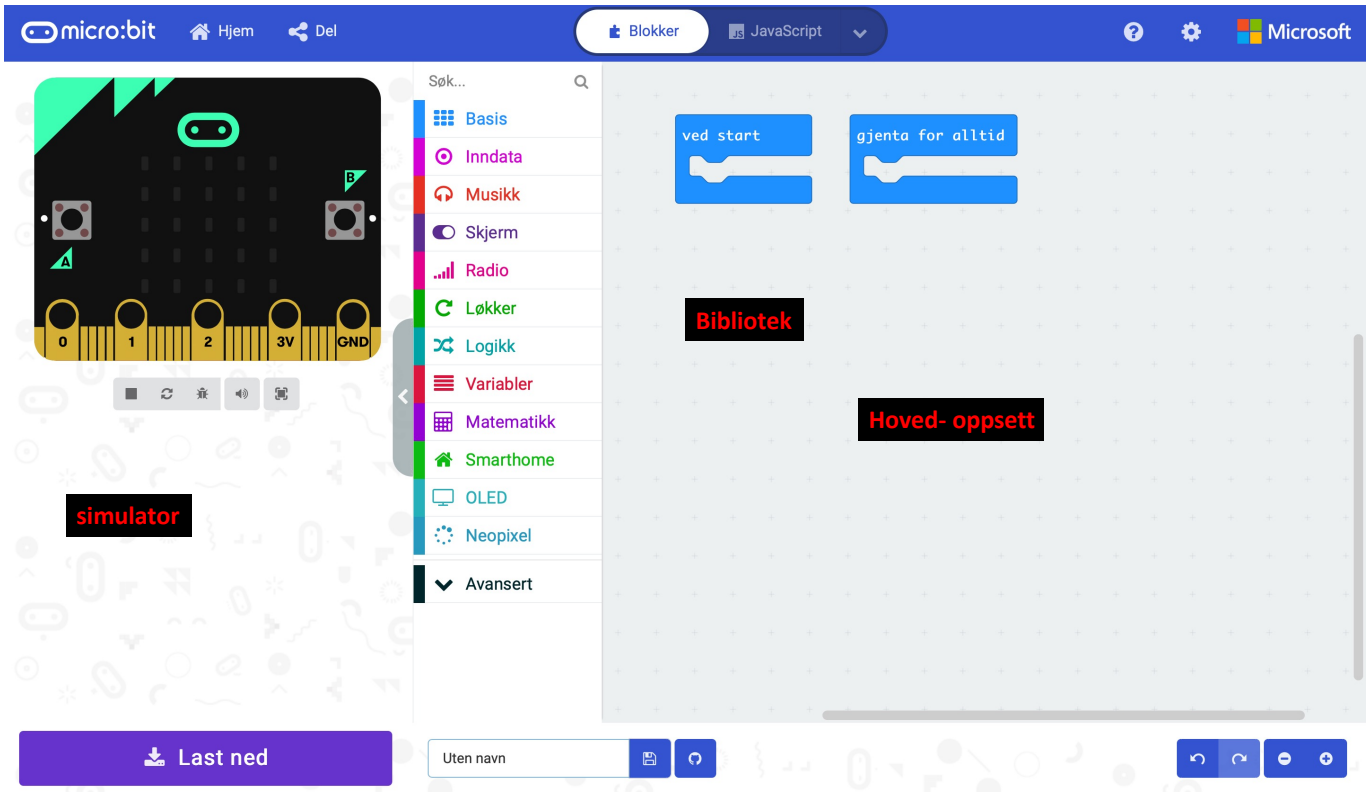
Breadboard / koplingsbrett



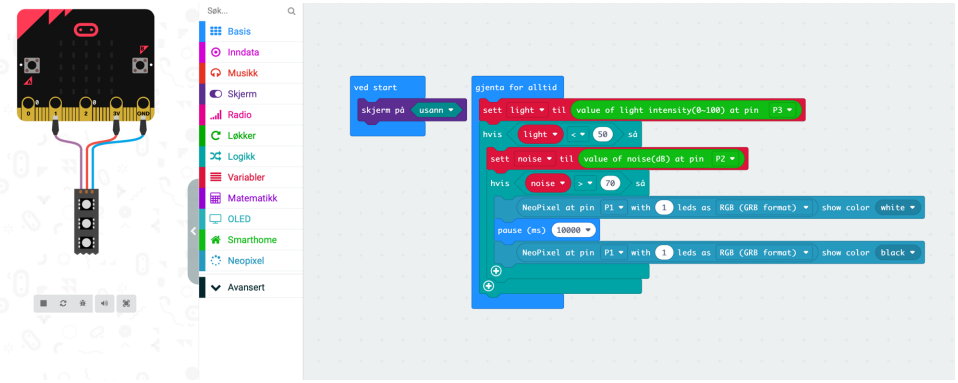




100Ohm Resistance
Adapter
Breadboard
Battery holder
Servo
Temperature Sensor
Photocell
USB
Jumper Wire
RGB LED
Red LED
Blue LED
Yellow LED
10k Trimpot
Pushbutton Switch
Self-lock Switch
Mini Speaker
10K Resistance
Clip Wire
5V Motor
Transistor
Diode
LED Ring


micro:bit
Smart home Kit
Specification


micro:bit
Hjem
Del
Blokker
JavaScript
Microsoft
Søk...
Basis
Inndata
Musikk
Skjerm
Radio
Løkker
Logikk
Variabler
Matematikk
Smarthome
OLED
Neopixel
Avansert
ved start
gjenta for alltid
Bibliotek
Hoved- oppsett
simulator
0
1
2
3V
GND
Last ned
Uten navn

Noise =0 og light2 = 0 beskriver at begge inngangssignalene starter på 0 før måling (LAV)
Led.enable(False) sette led til av-funksjon
Def on_forever(): indikerer at dette programmet skal løpe uendelig
Global light2, noise  I Python lar «global» deg endre variabelen utenfor gjeldende omfang. Den brukes til å lage en global variabel og gjøre endringer i variabelen i en lokal sammenheng, og vi slår fast at endringene innbefatter sensorene vi har navngitt (noise og Light2)
Light2 = smarthome.read_light_intensety(AnalogPin.P3) Her forteller vi hvilken føler som står på P3 og at verdien skal leses
If light2 < 50: hvis lysstyrken er mindre en 50
If noise > 70: hvis lyden er høyere en 70
Neopixel. Create(digitalPin.P1…. Da skal lampen lyse
Basic. Pause (10000) la den lyse i 10 sekunder (10.000 millisek.)
Neopixel. Create(digitalPin.P1… hvis ikke skal den lyse svar, dvs. ingen lys



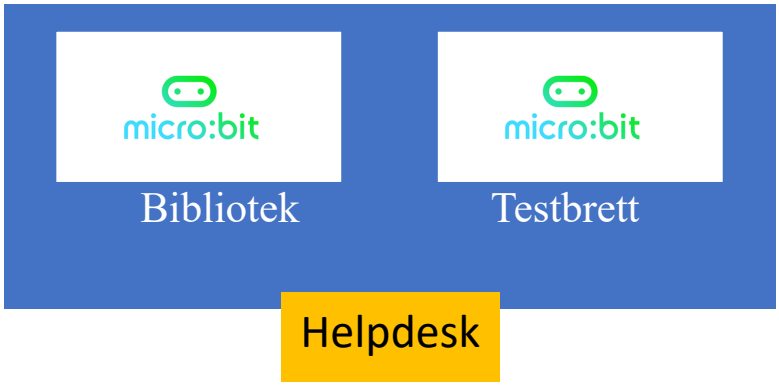

```
noise = 0
light2 = 0
led.enable(False)

def on_forever():
    global light2, noise
    light2 = smarthome.read_light_intensity(AnalogPin.P3)
    if light2 < 50:
        noise = smarthome.read_noise(AnalogPin.P2)
        if noise > 70:
            neopixel.create(DigitalPin.P1, 1, NeoPixelMode.RGB).show_color(neopixel.colors(NeoPixelColors.WHITE))
            basic.pause(10000)
            neopixel.create(DigitalPin.P1, 1, NeoPixelMode.RGB).show_color(neopixel.colors(NeoPixelColors.BLACK))
basic.forever(on_forever)
```

**Vi skal nå få utdelt materiell til noen gruppeoppgaver.**

Materiellet tildeles fra lærer ut fra hva vi har tilgjengelig i klassen / på skolen.

Gruppen velger ut fra listen og gjennomfører øvelsene som gruppe. Innlevering av øvelsene med tilhørende kommentarer er en del av karaktergrunnlaget.




## Sjekkpunkter, hva har jeg lært?

- Jeg er godt kjent med Micro: Bit, dens funksjoner og innebyggede muligheter
- Jeg kan kople opp grunnleggende porter i Boolsk Algebra
- Jeg kan laste ned og bruke IDE skjema til Micro: Bit, og kjenner hovedfunksjonene
- Jeg forstår hvordan man setter opp blokker i programmet og fordelen med disse
- Jeg kan sjekke programmet i utviklermiljø og laste det ned til Micro: Biten
- Jeg kjenner til forskjell på variabler og konstanter
- Jeg er kjent med if, if else, and, or, for og while logiske blokker og løkker
- Jeg vet hva et breadboard er og noen komponenters funksjon og virkemåte
- Jeg har gjennomført øvelsene for gruppen og levert inn

# Elevenes kunnskapsmål:

1: *Micro: Bit*

- Lær funksjonaliteten til MakeCode-programmeringsmiljøet og micro:bit-plattformen
- Bruke design-tenkningsprosessen til å utvikle en forståelse av et problem eller brukerbehov, og designe en optimal løsning
- Utøv kreativitet, «ingeniørarbeid» og tilgjengelige resurser for å lage, teste et prosjekt sammen med medelever.

*2: Algoritmer*

- Forstå de fire komponentene som utgjør en datamaskin og deres funksjoner
- Lær de forskjellige inngangene som micro:bit har, hvordan den behandler for å så sende informasjon til utganger
- Bruk denne kunnskapen og lag et micro:bit-program som har inndata og sender disse til utgang

*3: Variabler*

- Forstå hva variabler er, og når du skal bruke dem i et program
- Lær hvordan du oppretter en variabel, setter en startverdi og endrer denne
- Lær hvordan du oppretter meningsfulle variabelnavn
- Lære hvordan du bruker de grunnleggende matematiske blokkene for variabelverdier
- Opprette et program som bruker variabler

*4: Betingelser*

- Forstå hva betingede setninger er, og hvordan du bruker dem i et program.
- Finn ut hvordan du bruker de Logiske-blokkene Hvis... da' og 'Hvis ... da... ellers' for å gi spesifiserte resultater
- Opprette et micro:bit-spill som bruker betingede data riktig og effektivt

*5: Gjentakelse*

- Forstå verdien av gjentakelser i programmering
- Lær hvordan og når du skal bruke løkkeblokkene 'repeat', 'while' og 'for'
- Opprette et program som bruker gjentakelse og løkker

*6: Koordinater*

- Forstå sammenhengen mellom koordinater og de 5 x 5 LED´ene på micro:bit
- Forstå at verdiene i x - og y -koordinatene, hvordan du refererer til, tegner inn, opphever og veksler mellom dem
- Opprette et program som bruker koordinater, skattejakt

*7: Boolsk algebra*

- Forstå hva boolsk algebra og boolske operatorer er, og hvorfor og når de skal brukes
- Lær hvordan du oppretter en boolsk verdi, angir og endrer verdien
- Finn ut hvordan du bruker den tilfeldige sanne eller usanne blokken
- Opprette et program som bruker boolske operatorer

### *8: Biter, byte og binære funksjoner*

- Forstå hva biter og byte er, og hvordan de er relatert til datamaskiner
- Lær å telle i grunntall 2 (binærtall) og oversette tall fra grunntall-10 (desimal) til binær og desimaltall
- Opprette et program som bruker binær opptelling

### *9: Radiokommunikasjon*

- Forstå hvordan du bruker radioblokkene til å sende og motta data mellom to micro:bits
- Forstå de spesifikke datatypene som kan sendes over radioen
- Arbeid parvis for å designe et program ved hjelp av radiokommunikasjon mellom to mikro:biter

### *10: Fritt uavhengige prosjekter etter eget valg*

- Kode et eget prosjekt og utforme, programmere og bygge en fysisk funksjon, løsning eller hjelpemiddel som bruker Micro: Bit.