

Programming

- Arduino C/C++



Innholdsfortegnelse

Innledning	6
Hva er Arduino og hvordan fungerer den.....	7
<i>Ulike typer Arduino boards</i>	<i>10</i>
<i>Hvordan fungerer Arduino Uno-brettet og hvorfor det er så populært</i>	<i>11</i>
<i>Hva er en mikrokontroller</i>	<i>12</i>
<i>Hvordan bruke Arduino IDE (Integrated Development Environment).....</i>	<i>14</i>
<i>Link til Programvare nedlast</i>	<i>14</i>
<i>Slå på og koble Arduino til datamaskinen.....</i>	<i>15</i>
<i>Arduino API (Arduino Programming Language) er bygget opp på 3 hovedplattformer:</i>	<i>16</i>
<i>Link: Arduino API.....</i>	<i>16</i>
<i>Inn og utganger for Arduino UNO.....</i>	<i>16</i>
<i>Tekniske spesifikasjoner.....</i>	<i>17</i>
<i>Laste opp programmer til Arduino-brettet</i>	<i>18</i>
Best praksis for programmering!	19

Programmering Del 3

Øvelse: LED blink internt på Arduino kortet.....	20
Øvelse: La oss nå gjøre samme øvelse, men nå bruker vi Breadboard og digitale porter	23
<i>Fargekode kart for motstander</i>	24
<i>Kople opp ny krets etter tegning</i>	25
Link til kode	25
Litt teori og status så langt!	28
<i>Initiering av dataoverføring til PC</i>	29
<i>Bruk av variabler</i>	30
<i>Deklarasjon av lokale og globale variable</i>	31
<i>Kortets inn og utganger spesifisert</i>	32
<i>Tilleggskort</i>	33
Øvelse: De analoge portene.....	34
Link til kode	34
Link til kode	35
<i>Litt informasjon</i>	36
Øvelse beskrivelse av funksjon.....	37
Øvelse: Arduino og seriell port.....	38
Link til kode	38
Sjekkpunkter, hva har jeg lært?	42
Beslutningstaking og bruk av logikk.....	43
<i>If setning</i>	44

Programmering Del 3

Link til kode	50
.....	50
<i>While løkker</i>	51
Link til kode	54
<i>For løkker</i>	55
<i>Switch case løkken</i>	61
KORT OPPSUMERT	63
Link til kode	63
<i>Matematiske funksjoner</i>	65
Link til kode	69
<i>Opprette egne funksjoner</i>	70
Link til kode	77
Sjekkpunkter, hva har jeg lært?	78
Modul 4: Datamanipulasjon og EEPROM	79
.....	85
<i>Rekker/matriser</i>	86
Link til kode	90
<i>Operatører</i>	91
Aritmetiske operatører	91
Relasjonsoperatører.....	92
Logiske operatører	94

Programmering Del 3

Tildelings operatører	95
<i>BIT- MATEMATIKK</i>	96
Link til kode	111
<i>EEPROM</i>	112
Link til kode	117
<i>Bibliotek, serielle data og forskjellig tillegg</i>	118
SPI Interface	119
I²-C Interface	123
<i>Det finnes to typer busser seriell og parallell.</i>	124
Link til oppkopling	124
Link til kode	124
Bruk av interrupt og polling i Arduino program.	131
Link til kode	135
Diverse linker	135
Innhold Arduino sett	136
Sjekkpunkter, hva har jeg lært?	138

Innledning

I denne delen av programmeringen vil vi arbeide med programmeringsspråket C# / C++. For å kunne konkretisere opplæringen benytter vi Arduino Uno- mikrokontroller brett med tilhørende komponenter og en IDE (Integrated Development Environment) Platform med tilhørende biblioteker for Arduino.

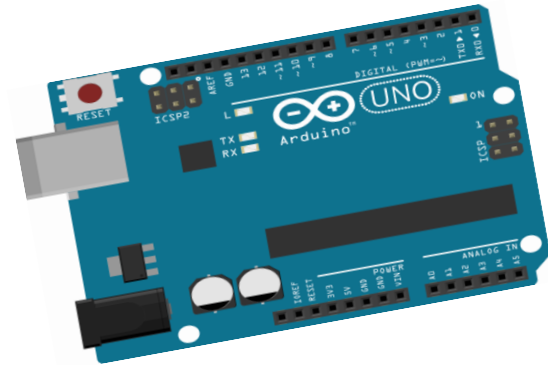
Noe av det viktigste i denne delen av progresjonen, er at man tar tiden til hjelp og er nøyaktig. Samtidig oppfordres det på det sterkeste at man skriver ned alle kodene selv og gjør egne merknader og har system på lagringen og mappestrukturen.

Husk også at man bruker tid på å bli en god programmerer, «ofte sier man at det vokser på deg»- som er et dumt uttrykk, da det i prinsippet betyr at man må bruke tid på det og ikke minst ha interesse for det. 2-3 timer i uka tar deg langt, noen timer om dagen tar deg til andre mål.

Vi skal i prinsippet berøre overflaten av mulighetene i denne delen og forsøke å vise til konkrete bruksområder og forhåpentligvis danne grunnlag for konseptutvikling i fremtiden.

Hva er Arduino og hvordan fungerer den

- Hva er Arduino?
- Ulike typer Arduino boards
- Hvordan fungerer Arduino Uno-brettet og hvorfor det er så populært?
- Hva er en mikrokontroller?
- Å9åHvordan bruke Arduino IDE (Integrated Development Environment)?
- Slå på og koble Arduino til datamaskinen
- Arduino API, Innganger og Utganger
- Laste opp programmer til Arduino-brettet



Ved slutten av dette kapitlet, vil du ha lastet opp ditt første program til Arduino UNO enheten for å kontrollere en LED. Vi kaller dette «Hello World» oppsett.

Arduino er en liten ” datamaskin ” som kan registrere og styre fysiske omgivelser og komponenter ved at man kobler den til ulike sensorer og styringsenheter.

Dette gjør Arduino til en velegnet enhet for å lære programmering og styring for både nybegynnere og mer avansert bruk/nivå. Programmering trenger ikke å være begrenset til apper som vises på en skjerm.

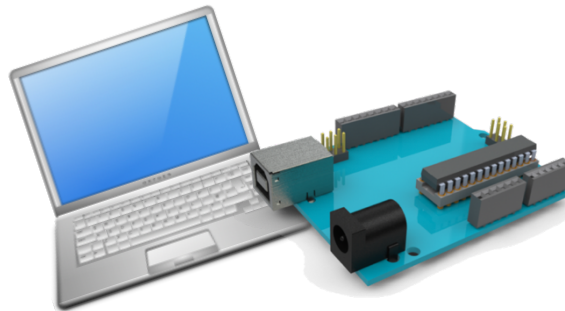
I dag finnes det koder som programmerere har skrevet, i det meste av utstyr som kobles til strømmettet eller drives av batteri.

- Mikrobølgeovnen har en timer- programvare som styrer hvor lenge maten skal varmes.
- TV-fjernkontrollen har programvare med kommandoer som sendes når noen trykker på en knapp.
- Robotstøvsugeren kjører en avansert programvare som styrer hvordan støvsugeren beveger seg, kommer seg forbi hindre og finner tilbake til ladestasjonen.
- Boligene våre styres mer og mer av elektronikk og kontrollenheter, såkalte sensorer og alt fra enkle til avanserte programmer, IoT (Internet of Things).

Programmering Del 3

Arduino er løsningen som gjør at alle, både nybegynnere og erfarne programmerere kan skrive kode som kjøres utenfor mobilens, nettbrettets eller datamaskinens skjerm. Med Arduino kan alle som har interesse av det, bygge egne elektronikkprosjekter. Det kan dreie seg om alt fra lysdetektorer og vekkerklokker til mobiltelefoner og radiostyrte bilder.

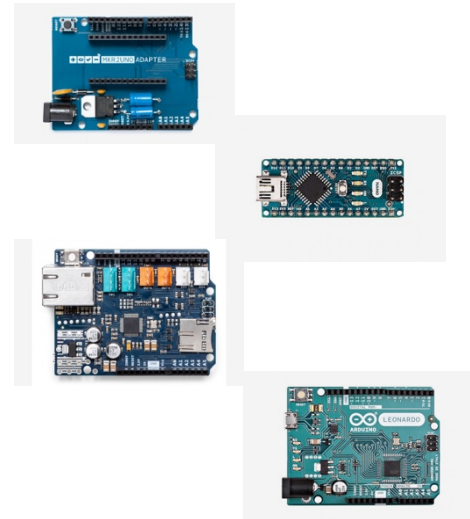
I slutten av denne modulen ligger det linker til en rekke prosjekter og butikker hvor man kan kjøpe inn komponenter og sensorer, inkludert våre egne systemoppsett som dekker modulene i våre kurshefter.



Ulike typer Arduino boards

Det finnes en rekke ulike Arduino kort, noen forskjellige størrelser av selve programmkortet og en rekke tilleggskort innenfor motorstyring, relekontroller, Ethernet-modul og liknende.

Her har vi bilder av noen av dem.



I denne delen kommer vi til å arbeide med Arduino UNO. I del 4 og del 5 tar vi for oss noen av de andre modulkortene og styringene for å vise disse i funksjon også.

Hvordan fungerer Arduino Uno-brettet og hvorfor det er så populært

Arduino Uno er et mikrokontrollerkort basert på ATmega328P ([dataark](#)). Den har 14 digitale inngangs- / utgangspinner (hvorav 6 kan brukes som PWM-utganger), 6 analoge innganger, en 16 MHz keramisk resonator (CSTCE16M0V53-R0), en USB-tilkobling, en strømkontakt, en ICSP-header og en tilbakestillingsknapp.

Den inneholder alt som trengs for å støtte mikrokontrolleren; bare koble den til en datamaskin med en USB-kabel eller kople den til med en AC-er du i gang.

Du kan arbeide med Arduino gjøre noe galt, i verste fall, om brikken for noen få kroner og



Uno uten å bekymre deg for å uhellet er ute kan du erstatte starte på nytt.

"Uno" betyr **EN** på italiensk og ble valgt for å markere utgivelsen av Arduino Software (IDE) 1.0. Uno-brettet og versjon 1.0 av Arduino Software (IDE) var referanseversjonene av Arduino, nå utviklet seg til nyere utgivelser. Uno-kortet er det første i en serie USB baserte Arduino-brett, og referansemodellen for Arduino-plattformen; for en totaloversikt over nåværende kort og løsninger gå til [Arduino.cc](https://www.arduino.cc)

Hva er en mikrokontroller

En mikrokontroller (μC) er en integrert chip som ofte er en del av et innebygd system. Den inneholder en CPU, RAM, ROM, I/O-porter, mm. som en vanlig datamaskin. Fordi de er laget for å utføre kun en bestemt oppgave og å styre et enkelt system, er de mye mindre og forenklet slik at de inkluderer alle funksjonene som kreves på en enkelt brikke.

En mikroprosessor (μP), som er en mer generell chip som brukes til å lage en Multifunksjons datamaskin eller enhet, og krever flere chips for å kunne håndtere ulike oppgaver.

Mikrokontrollere er ment å være mer selvstendige og uavhengige og brukes mye i såkalte Embedded systemer.

Mikrokontrolleren inneholder regne- og data-manipuleringsenheten, lager for program og data og det som skal til for å kommunisere med omverdenen, enten det er analoge spenninger eller digitale inn- og utganger, så finnes alt dette inne i den integrerte kretsen.

Mikrokontrolleren er med andre ord en komplett datamaskin inkludert i en integrert krets.

Kort oppsummert

- Arduino er en mikrokontroller
- En mikrokontroller er en integrert datamaskin på en chip

Programmering Del 3

Prisen en må betale er at den ikke er så kraftig og generell som en mikroprosessor, men som oftest er den kraftig nok til vårt bruk. Den mangler dessuten muligheter for direkte å koble til PC-tastatur og skjerm.

Skal vi kommunisere med en mikrokontroller er vi derfor som oftest avheng av å bruke en PC med tilpasset programvare for å programmere den. Når den først er programmert kan den frigjøres fra tastatur, mus, skjerm og PC.



Arduino kan programmeres til å utføre akkurat den oppgaven (eller oppgavene) som programmereren vil. Det fine med Arduino er at alt er basert på åpen kildekode. Hvem som helst kan bruke og forbedre maskinvaren og programvaren. Ved hjelp av Arduino kan man for eksempel bygge en robotgressklipper, en dørlås med RFID, en automatisk plante vanner eller en GPS-sporer, osv.

Mulighetene er ubegrensede!

Hvordan bruke Arduino IDE (Integrated Development Environment)

Downloads



Arduino IDE 1.8.13

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

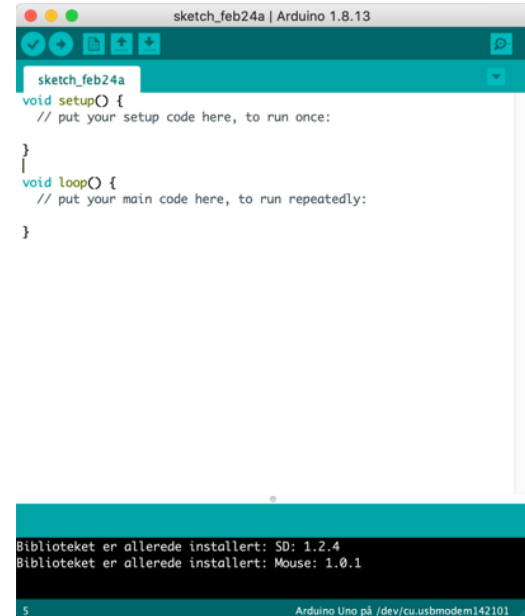
Refer to the [Getting Started](#) page for installation instructions.

SOURCE CODE
Active development of the Arduino software is [hosted by GitHub](#). See the instructions for [building the code](#). Latest release source code archives are available [here](#). The archives are PGP-signed so they can be verified using [this](#) gpg key.

DOWNLOAD OPTIONS

- Windows** Win 7 and newer
- Windows** ZIP file
- Windows app** Win 8.1 or 10 [Get](#)
- Linux** 32 bits
- Linux** 64 bits
- Linux** ARM 32 bits
- Linux** ARM 64 bits
- Mac OS X** 10.10 or newer

Release Notes: Checksums (SHA512)



sketch_feb24a | Arduino 1.8.13

```
sketch_feb24a
void setup() {
  // put your setup code here, to run once:
}
void loop() {
  // put your main code here, to run repeatedly:
}
```

Biblioteket er allerede installert: SD: 1.2.4
Biblioteket er allerede installert: Mouse: 1.0.1

Arduino Uno på /dev/cu.usbmodem142101

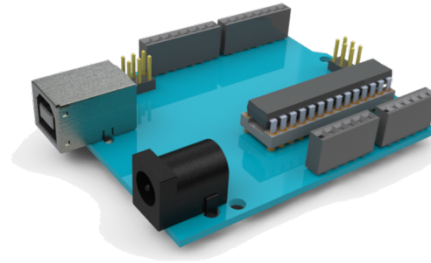
Link til Programvare nedlast

Sjekk at du laster ned riktig programvare til din maskintype og siste versjon. Det er lurt å lage din egen Arduino mappe, slik at du har programvaren og alle lagrede øvelser samlet på en plass. En god mappestruktur og navn på filene dine som beskriver øvelsene, gjør det lett å finne frem til senere.

Slå på og koble Arduino til datamaskinen



USB kabel type A-B



Du trenger følgende for å kople opp maskin og kort. En PC/MAC, USB kabel type A-B og en Arduino UNO. På datamaskinen din må du ha lastet ned IDE og ha Sketch lastet ned og klar til bruk.

Kople opp alle enhetene og du er klar til første lille test for å kontrollere at vi har kontakt. Dette kalles normalt «Hello World» eksperimentet og har flere oppgaver:

- Vi har lastet ned riktig program
- Kabelen fungerer
- Arduinoen fungerer
- Og vi får lastet ned og startet programmet på Arduinoen

Arduino API (Arduino Programming Language) er bygget opp på 3 hovedplattformer:

- Funksjoner
- Variabler
- Struktur

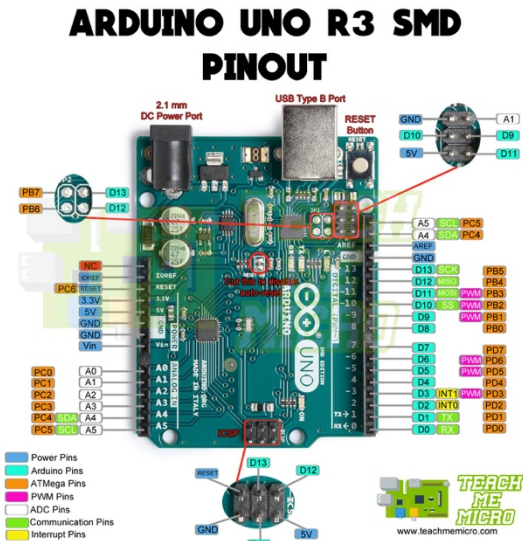
Har du lastet ned [Arduino IDE](#)

[Link: Arduino API](#)

Ved hjelp av lenken på foregående side kan du gå inn på Arduino.cc sin hjemmeside. Alle 3 API plattformene har linker til forklaring på bruk og definisjoner av de ulike operandene.

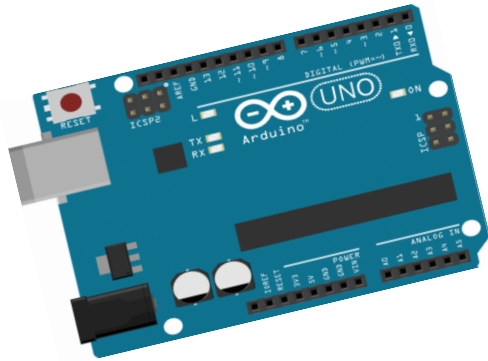
Inn og utganger for Arduino UNO

Utviklingskortet Arduino Uno har 14 digitale inn- og utganger, og disse kalles IO- (Input Output) eller GPIO-pinner (General Purpose Input Output). IO-pinnene brukes til å koble til digitalt tilbehør, for eksempel lysdioder og trykknapper.



Tekniske spesifikasjoner

Programmering Del 3



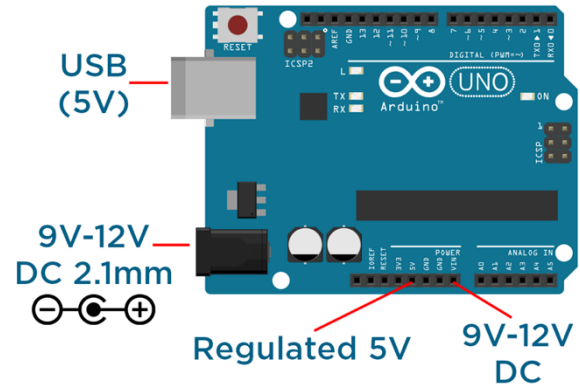
Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Laste opp programmer til Arduino-brettet

I denne delen lærer du om de ulike måtene å drive Arduinoen på, og hvordan du kobler den til datamaskinen for å laste opp programmene dine, og kommunisere til datamaskinen ved hjelp av den serielle porten.

Den letteste måten å drive Arduino på er ved å bruke USB porten, denne kan levere både strøm og kommunikasjon for opplasting av programmer. Du kan også forsyne den med 5-12V gjennom oppsatte strømplugg (rund). Det er også satt opp direkte tilkoplinger for 5V og 9 til 12V, som går utenom regulatoren, så her må man være forsiktig.

USB er den sikreste tilkoplingen for strøm og for kommunikasjon, spesielt under prototypeoppsett og testing. Senere kan man sette opp egen forsyning med eksempelvis batteri eller nettilkopling.



Best praksis for programmering!

En god struktur under programmering er viktig og innrykkene og bruk av store bokstaver i delte ord er ikke bare til pynt.

Innrykkene gir oversikt over hvilke kodegrupper som hører sammen (familiære) og hvilke nivåer de ulike kodene ligger på. Store bokstaver i delingsord, slik som **pinMode**, et sammensatt ord som beskriver en inn/utgangs status.

Denne formen for struktur gir bedre oversikt. Semikolon som benyttes etter hver linje for å gi kommandoen **UTFØR**. Logiske uttrykk og definisjoner bruker ikke semikolon, da disse egentlig ikke forteller mikrokontrolleren om å utføre en instruksjon eller aktivitet, den kontrollerer bare flyten i koden din.

```
code-layout-example
1 void setup() {
2   pinMode(13, OUTPUT);
3 }
4
5 void loop() {
6   digitalWrite(13, HIGH); //note the indents created using 'TAB', de-indent using 'SHIFT + TAB'
7   delay(500); //keep all code spaced to the same indent for each nested level
8   digitalWrite(13, LOW); //double forward slash allows you to create comments which aren't compiled
9   delay(500); //these help organise your code and increase readability
10 }
11
12 void sampleFunction() //brackets should be spaced as above, or as below
13 {
14   //code goes here
15 }
```

Øvelse: LED blink internt på Arduino kortet

Husk å lagre øvelsene i din egen mappe, dette er første øvelse på kommunikasjon med Arduino UNO gjennom din egen datamaskin

[Link til bibliotek og forklaring](#)

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

Legg merke til hvordan man beskriver funksjonene ved hjelp av // og tekst eller avsnitt start /* avsnitt slutt */

Programmering Del 3

Åpne utviklingsmiljøet til Arduino, Arduino IDE, og dette bildet vises (Vi kaller arbeidsområde for Sketch område)

1. Gå til fil og velg Examples (Fil er normalt pil opp)
2. Velg 01Basics og blink
3. Et tekstvindu åpner seg, la dette ligge foreløpig
4. Velg Tools og Board- velg det brettet du har fått utdelt (Sannsynligvis Arduino UNO)
5. Velg Tools og serieport (i Windows normalt den høyest nummererte porten)
6. Er du usikker på hvilken port, kan du kople fra Arduino kortet og se hvilken som forsvinner fra Com oppsettet, kople til igjen og velg denne.



```
sketch_feb24a | Arduino 1.8.13
sketch_feb24a
void setup() {
  // put your setup code here, to run once:
}
void loop() {
```



```
sketch_feb24a | Arduino 1.8.13
sketch_feb24a
Blink | Arduino 1.8.13
Blink
//
Turns an LED on for one second, then off for one second, repeatedly.
Most Arduinos have an on-board LED you can control. On the UNO, the
correct LED pin independent of which board is used.
If you want to know what pin the on-board LED is connected to on your
model, check the Technical Specs of your board at:
https://www.arduino.cc/en/Main/Products
```



```
Blink | Arduino 1.8.13
Blink
```

For å laste opp programmet til Arduinoen din, velg Upload, dvs. pilen øverst til venstre. Husk å lagre filen, det er pilen lengst til høyre som peker nedover.

VIS PROGRAM EDITOR

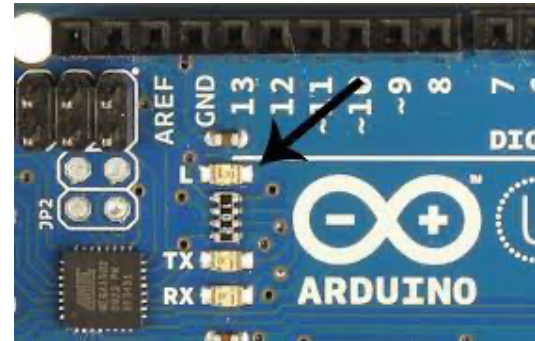
Programmering Del 3

Programmet lastes fra Pc til kort, som du vil se på nederst i IDE brettets sorte felt (Scetch området). TX og RX LED på brettet skal nå blinke, nå programmet lastes opp, og du får teksten Done Uploading i IDE Sketch området ditt når klart.

Etter noen få sekunder vil LED lampen merket L på Arduino UNO brettet starte å blinke med 1 sek. Mellomrom. Du har nå oppnådd kommunikasjon mellom PCen din og kortet.

MERK:

Av og til er Arduino satt opp med blinkefunksjonen fra fabrikk når de kontrollerer brettene. For å være sikker på at det er ditt program som er i funksjon på brettet (Kontroll) så kan du endre tidsrammene på delay low og delay High til 5000, større intervaller og laste ned på nytt.



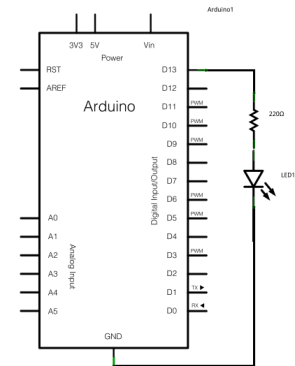
Øvelse: La oss nå gjøre samme øvelse, men nå bruker vi Breadboard og digitale porter

Eksempel på bruk av ekstern diode på Breadboard

```
void setup() {
  pinMode(13, OUTPUT);    //setup pin 13 as an output
}

void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH outputs 5V)
  delay(500);             // wait for 500 milliseconds
  digitalWrite(13, LOW);  // turn the LED off (LOW outputs 0V)
  delay(500);             // wait for 500 milliseconds
}
```

Når vi skal teste eksemplene, bør vi montere kretsene på et «Breadboard» og koble til **elektroniske komponenter**. Du får disse utdelt av den som underviser til hver øvelse eller et standard sett for alle øvelsene for denne modulen. Her ser vi oppkopling direkte på Arduinoen og det skjematiske oppsettet til en blinkende LED.

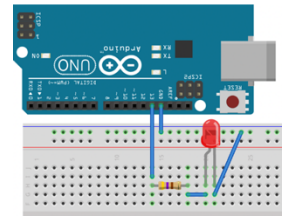
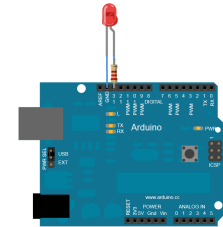


Programmering Del 3

Oppkoplingen består av en LED av valgt farge- rød, en motstand, og når vi skal kople dette opp på Breadboard, så trenger vi også noen kabler i ulike størrelser og farger.

NB! Kretsen vises uten bryter!

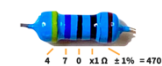
Når du skal gjøre denne øvelsen med ekstern LED, må du bygge denne kretsen. Her skal du koble den ene enden av motstanden til den digitale pin-utgangen *til LED_BUILTIN* konstant. Det lange benet på LED-lampen (det positive benet, kalt anoden) til den andre enden av motstanden. Koble det korte benet på LED-lampen (det negative benet, kalt katoden) til GND. I diagrammet ovenfor viser vi Arduino UNO- som har D13 som *LED_BUILTIN* verdi. Verdien av motstanden i serie med LED kan være av en annen verdi enn 220 ohm; lysdioden tennes også med verdier opp til 1K ohm, 1000 ohm.



Fargekode kart for motstander

Når man leser av verdiene på en motstand leses den fra venstre mot høyre. Noen motstander har 2 signifikante verdier, andre flere- så en multiplikator ring og en toleranse ring.

Color	Significant digits	Multiplier [Ω]	Tolerance [%]
Black	0	x1	—
Brown	1	x10	± 1
Red	2	x100	± 2
Orange	3	x1,000	—
Yellow	4	x10,000	—
Green	5	x100,000	± 0.5
Blue	6	x1,000,000	± 0.25
Violet	7	x10,000,000	± 0.10
Grey	8	x100,000,000	± 0.05
White	9	x1,000,000,000	—
Gold	—	—	± 5
Silver	—	—	± 10
None	—	—	± 20



Kople opp ny krets etter tegning.

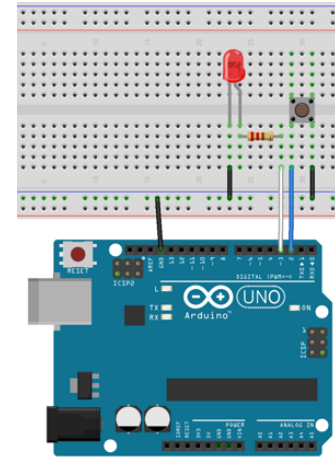
```

int buttonPin = 2;
int ledPin = 3;

void setup() {
  // setup pin modes
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
  // read state of buttonPin and store it as the buttonState variable
  int buttonState = digitalRead(buttonPin);
  // write the value of buttonState to ledPin
  digitalWrite(ledPin, buttonState);
}

```



[Link til kode](#)

Programmering Del 3

Oppkopling med bryter, aktiv lav logisk funksjon

La oss gå gjennom kodingen steg for steg! Innledningsvis er dette eksperimentet lagt opp med en digital Read og digital Write funksjon. Derfor ser du at vi kun benytter de digitale portene på Arduinoen vår.

Vi må sette opp 2 globale variabler i stedet for direkte knytting til inn og utganger.

`int buttonPin = 2;` angir at vi får et inngangssignal fra pinne 2, som angir om bryter har vært trykket på eller ikke. `int ledPin = 3;` gir et utgangssignal til pinne 3 basert på inngangssignal og programoppsett.

(Lokale variabler er kun for en gangs tilfeller, global betyr at hver gang man nevner variabelen, vet Arduinoen at man skal benytte aktuelle angitte funksjon/port)

`Void setup()` løper kun en gang, og bare en gang!

Vi skriver `PinMode` og henviser til den innledende variabelen `int ledPin` som igjen viser til port 3, som vi definerer til å bli en `OUTPUT`.

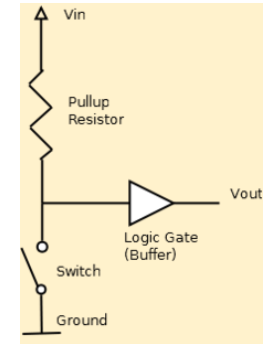
Vi skriver `PinMode` og henviser til den innledende variabelen `buttonPin` som igjen viser til port 2, som vi definerer til å bli en `INPUT_PULLUP`.

Programmering Del 3

INPUT_PULLUP er en spesiell definisjon. Når man definerer en vanlig **INPUT** funksjon på en inngang, er den ekstremt følsom og dermed ustabil, den kan eksempelvis reagere på små statiske endringer i luften.

Vi setter derfor inn en **PULLUP resistor/motstand** på eksempelvis 10K til 100K og kopler den til 5V signalet internt. (Motstanden er innebygget, men man kan også kople den opp om ønskelig på Breadboard).

Motstanden er så høy at det flyter så godt som ingen strøm gjennom, men nok til å holde lys i LEDen vår, den er koplet til 5V internt og defineres høy (1), når bryteren ikke er trykket på. Når bryteren trykkes inn blir signalet lavt (0), på grunn av den store motstanden! Ved hjelp av denne funksjonen får man en klart definert høy eller lav funksjon og ikke en flytende funksjon som er udefinert.



Void loop () løper helt til vi slår den av eller koder den til å gå av!

Vi skriver en lokal variabel `int buttonState = digitalRead (buttonPin);` leser status på det digitale signalet fra `buttonPin` som kommer fra port 2.

`digitalWrite(ledPin, buttonState);` sender utgangssignal (høy eller lav) ut fra signalet lagret i `button state` basert på signal fra `button pin`. Er bryter ikke trykket er den høy, er den trykket går den lav LED, funksjon styrt av eller på.

Kople opp, skriv inn koden og last ned og test funksjonen!

Litt teori og status så langt!

Å få en LED til å blinke avhengig av ulike oppkoplinger er i seg selv ikke så vanskelig, men i prinsippet har du allerede på dette tidspunktet klart noen enkle grunnleggende programmeringsoppsett for å kunne styre mange funksjoner i hverdagen. Når vi nå skal gå videre trenger vi litt informasjon om inn og utgangene, funksjonen i Arduino UNO og litt enkel komponentlære.

Setup () og loop () er *navnet* til funksjonene, mens de to klammeparentesene {} omslutter *kroppen til funksjonen* hvor selve programmet ligger.

I tillegg til disse to faste funksjonene så bruker vi funksjoner fra Arduino's eget bibliotek (kalt *Arduinodef.* på figuren), funksjoner som andre har laget (*Andrefdef.*)

og vi kan lage våre egne funksjoner (*Egendif.*). Funksjoner som andre har laget og som de ønsker å tilby fellesskapet, samles ofte i *biblioteker*, som kan lastes ned, installeres og gjøres tilgjengelig for programmerere.

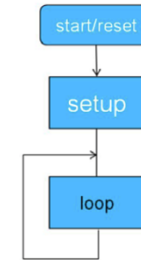
```
#include <bibliotek> // Inkluderer spesielle biblioteker
Deklarer globale variable

void setup()
{
  // Koden i setup() kjører bare ved start
  .... pinMode(<pin>, in/output);
}

void loop()
{
  // Deklarer lokale variable
  // Programlinjer

  funksjon1(); // Kaller funksjon 1
  ...
}

void funksjon1()
{
  // Deklarer lokale variabler
  // kode
}
```



Programmering Del 3

Referansemanualen til Arduino C++ for bruk ved programmering av Arduino-kontrollerkort finnes på følgende nettside: <http://arduino.cc/en/Reference/HomePage>

Initiering av dataoverføring til PC

Under uttestingen og visning av innhold av variabler kan det være praktisk at data leses tilbake til monitoren i Arduino-editoren. Datahastigheten settes opp i void setup()funksjonen med kommandoen: `Serial.begin(9600);` her er datahastigheten satt til 9 600 baud (her ca. 9600 *bit* pr. sekund) som vist under:

```
void setup()
{
  Serial.begin(9600); }
```

I noen tilfeller kan det være hensiktsmessig å øke overføringshastigheten. 115 200 tegn pr. sekund er også vanlig å bruke. Pass på at mottakerhastigheten for monitoren er satt til samme hastighet. Dette gjøres nederst til høyre i monitoren som vist på figuren her.

Baud rate - Angir hvor mange symboler som kan overføres pr. transmisjonsperiode. For et binært system vil baud-raten tilsvare bit pr. sek. For mer komplekse former for modulasjon (f.eks. QPSK) vil hver transmisjon periode kunne overføre flere bit (2, 4 eller flere), dog på bekostning av økt følsomhet for støy.



Bruk av variabler

Bruk av variabler er en praksis som gjør programmering særdeles slagkraftig.

Vi kan betrakte variabler som oppbevaringssteder (” skuffer”) for tekst eller tallverdier med navn og type. Innholdet vil som oftest være ukjent når vi skriver programmet, men vi reserve- rer plass til verdiene. Navnene på variablene bør gjenspeile hva de representerer slik at det blir lettere og lese og forstå koden.

La oss deklare variablene” tempForskjell”, ”tempStart” og ”tempSlutt” som float (desimaltall). Vi kommer da til den andre viktige egenskapen:

Vi kan behandle og utføre beregninger på variablene uten at verdiene til variablene er kjente.

```
tempForskjell = tempSlutt – tempStart;
```

Som variabelnavnene indikerer, så ønsker vi å beregne temperaturforskjellen mellom start- og sluttidspunktet. Dette føres oss til den tredje viktige egenskapen:

Vi kan bruke variabler som lagringsplass for verdier over tid.

Deklarasjon av lokale og globale variable

I programspråkene C og C++ må alle variabler deklarerer før de kan brukes. Deklarasjonene må inneholde type og navn på variabelen og gjøres gjerne i starten av programmet før void setup()-funksjonen. Variabler deklartert utenfor funksjonene blir **globale variabler**, dvs. at variablene kan brukes i alle funksjonene og beholder innholdet uavhengig av hvor de brukes.

Deklarering av variabler kan også gjøres innenfor hver funksjon. Slike variabler gjelder da bare innenfor denne funksjonen og kalles **lokale variabler**. Under er vist deklarasjon av de vanligste typer variabler. I dette tilfelle vil de bare gjelde innen funksjonen void loop():

```
void loop() {
```

- `Int a;` // deklarasjon av 16 bit heltallsvariabel (word)
- `char b;` // deklarasjon av 8 bit karaktervariabel (byte)
- `char c, d;` // deklarasjon av to 8 bits karaktervariabler (byte)
- `float e;` // deklarasjon av variabelen e som et desimaltall f.eks. 1,65
// (32 bit, dobbel word)
- `unsigned long f;` // deklarasjon av 4 byts heltallsvariabel f (32 bit)
// uten fortegn
- `boolean g;` // deklarasjon av en boolsk variabel g
// som kan ha verdiene 0 eller 1
// Her følger resten av programkoden i funksjonen loop()-

```
}  
Dersom vi deklarerer variabler i begynnelsen av loop()-funksjonen så vil de bli redeklarerert for hver runde i loopen, hvilket betyr at vi må regne med at de vil miste den verdien de har.
```

Kortets inn og utganger spesifisert

- **Digitale I/O-porter**

Kortet har 14 digitale inn/utporter

(I/O-porter) som kan programmeres til enten å være en inn- eller en utgang. Seks av disse (3, 5, 6, 9, 10 og 11) kan *pulsbreddemoduleres* (pwm), disse er merket med ~ på kortet. Maksimal strøm på hver av I/O portene er 40 mA.

- **Analoge innganger**

Kortet har 6 analoge innganger som kan tilføres spenninger fra 0 – 5V. Spenninger ut over 5V vil ødelegge mikrokontrolleren. Disse kan også programmeres om slik at de kan brukes som digitale inn og utganger.

- **USB-kontakt** for direkte tilkobling av PC og for programmering av kortet og overføring. I tillegg kan data overføres mellom monitoren på PC-en og kortet. Under programmeringen tilføres kortet spenning fra USB-kontakten. Dersom denne belastes med mer enn 500 mA vil strømforsyningen bli brutt inntil strømtrekket reduseres under denne grensen. USB3 kan levere vesentlig høyere strøm.

- **Strømtilførsel**

Tilkoblingspunkter for batterieliminatør anbefalt spenning 7 – 12 V (grenseverdier 6 – 20 V). Batteri kan enten tilkobles eliminatorpluggen (2.1 mm + senter) eller via V_{in} (+) og GND (–). Enkelte komponenter trenger lavere spenning som ev. kan leveres fra 3.3 V utgangen på kortet.

- **Reset**

Kortet inneholder en RESET-knapp som resetter programmet og starter det på nytt.

Tilleggskort

Kantkontaktene er montert slik at ulike tilleggskort (“shield card”) kan monteres rett ned på Arduino-kortet.

Kortet støtter ulike typer datakommunikasjon med omverdenen: UART (Rx/Tx), I²C-databuss og SPI-databuss.

[For den interesserte så kan kretsskjema for Arduino UNO](#)

[For den interesserte så kan oversikt over elektriske symboler hentes her](#)

Øvelse: De analoge portene

Kildekode for blink kontroll

```

int ledPin = 3;
int potPin = A0;

void setup() {
  // setup pin modes
  pinMode(ledPin, OUTPUT);
  pinMode(potPin, INPUT);
}

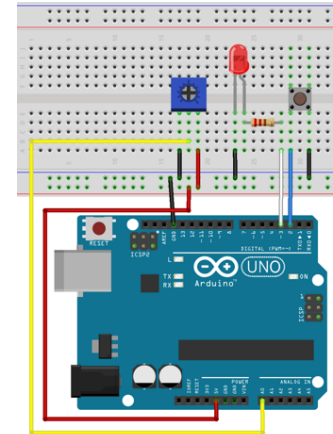
void loop() {
  // read the value of the pot and store it as potValue
  int potValue = analogRead(potPin);

  // turn led on and wait for the time equal to potValue
  digitalWrite(ledPin, HIGH);
  delay(potValue);

  // re-read the value of the pot and store it as potValue
  potValue = analogRead(potPin);

  // turn led off and wait for the time equal to potValue
  digitalWrite(ledPin, LOW);
  delay(potValue);
}

```



[Link til kode](#)

Kildekode for justerbart lys

```
int ledPin = 3;
int potPin = A0;

void setup() {
  // setup pin modes
  pinMode(ledPin, OUTPUT);
  pinMode(potPin, INPUT);
}

void loop() {
  // read the value of the pot, divide it by 4, and store it as potValue
  int potValue = analogRead(potPin) / 4;

  // turn led on with the value of potValue
  analogWrite(ledPin, potValue);
}
```

[Link til kode](#)

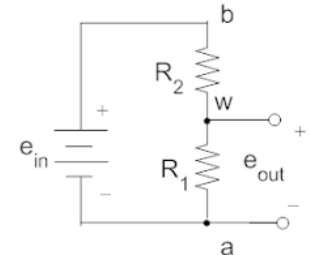
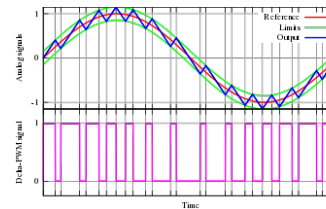
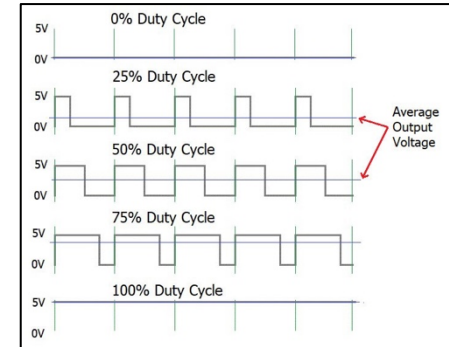
Litt informasjon

5V oppdelt i en 10 bits struktur, 0-1023(1024 muligheter) viser til sampling forholdet, og dermed nøyaktighetene i AD konverteren (Analog til digital konverter). Det betyr i praksis at vi har 10 bit analog og 8 bit digital ($1024/4 = 256$).

Man vil fort kunne tenke at det analoge signalet som sendes ut, Analog Write, er en variabel spenningsstyrke. I virkeligheten gjør den ikke det, den sender ut et digitalt PWM (puls brede modulasjon) signal, som sender ut hundre/tusenvis, av og på signaler til porten, som da slår LED lampen av og på så raskt at vi ikke oppfatter det med det menneskelige øye.

Man kunne montert inn en DAC-konverter og gjort signalet om til et analogt signal, men i mange tilfeller er dette ikke nødvendig og medfører ekstra kostnader. Man kunne også lagt det inn i programmet, men dette ville krevd mye unødig programmering og resurser av mikrokontrolleren vår.

Pulse Width Modulation, eller PWM, er en teknikk for å få analoge signaler ut av digitale enheter. Digitale medier brukes til å lage en firkantbølge, et signal som arbeider mellom av og på (Høy og lav).



Programmering Del 3

Som vi ser av bildet til høyre, kan man regulere av og på tiden, og ved hjelp av denne funksjonen og dermed gjennomsnittlig spenningsnivå, som igjen regulerer styrken på lyset i LED lampen.

I denne øvelsen bruker vi også et potensiometer, som skal fungere som en spenningsdeler.

Øvelse beskrivelse av funksjon

- a. Blink kontroll med analoge porter
- b. Justerbart lys med analoge porter

Denne øvelsen krever at man setter seg inn i virkemåten og ser for seg hvordan utvikleren har tenkt at programmet skal fungere. En god utvikler må kunne lese funksjoner, men fortvil ikke, dette er første steget på veien og lykkes du ikke, så går vi selvsagt gjennom disse funksjonen i detalj. Det fint å arbeide sammen 2 og 2 også.

Øvelse: Arduino og seriell port

Kildekode for seriell monitor via serieporten

```
int ledPin = 3;
int buttonPin = 2;
int potPin = A0;

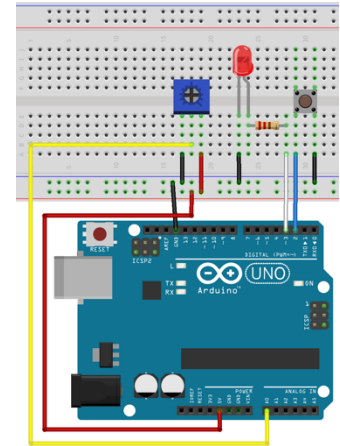
void setup() {
  // setup pin modes
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(potPin, INPUT);

  // initialise serial port with baud rate of 9600
  Serial.begin(9600);
}

void loop() {
  // read the state of buttonPin and store it as buttonState (0 or 1)
  int buttonState = digitalRead(buttonPin);

  // read the value of the pot, divide it by 4, and store it as potValue
  int potValue = analogRead(potPin);
  int filteredPotValue = potValue / 4;

  // turn led on with the value of potValue
```



Link til kode

Programmering Del 3

```
analogWrite(ledPin,filteredPotValue);

// print the value of the button
Serial.print("Button: ");
Serial.print(buttonState);
Serial.print(" ");

// print the value of the pot
Serial.print("Pot: ");
Serial.print(potValue);
Serial.print(" ");

// print the value of the pot / 4 with a line return at the end
Serial.print("Pot/4: ");
Serial.println(filteredPotValue);
}
```

Nå vil vi se på hvordan vi kan bruke den serielle porten på Arduino IDE for å kommunisere med en datamaskin via USB. På denne måten kan vi få en innsikt i hvordan Arduinoen sin Microprosessor arbeider. Denne metoden er en viktig funksjon og gir oss detaljert informasjon om hvordan prosessoren arbeider og hvordan ulike sensorer avgir informasjon som vi senere kan bruke til ulike oppgaver, eksempelvis til værstasjon og snittverdier m.m.

[Link til seriell port forklaring](#)

Vi legger inn kildekoden for seriell monitor og tar en kikk på hvordan monitoren og programmet fungerer.

Innledningsvis ser vi de globale variablene, int. ledPin, buttonPin og potPin angir inn og utganger, og void setup () som initierer pin mode (inn og utganger), og legg også her merke til Input Pullup for å stabilisere signalene (Høy og Lav). Serial.begin(9600); er Arduinoen sin tilkopling til det serielle biblioteket (TX/RX, og 9600 som baud rate er antall bits pr. sek.

Void loop int buttonState=digital read (buttonPin), les av status på port 2 som gir deg status på bryteren, som er enten av eller på (0 eller 1).

int potValue = analogRead(potPin); leser av verdien på inngang A0 og filtrerer denne med int filteredPotValue = potValue / 4; Så hvorfor deler vi på 4?

Arduino sin innebyggede DAC har en 10 bits konverter, men en 8-bit-oppløsning. $0+1023$ (1024) muligheter / 4 er 256 eller $0 + 255$ som er en 8 bits oppløsning.

Arduino ADC Specification

Parameter	Arduino Uno/Nano
Voltage Supply (Vs)	1V8 ~ 5V5
Interface	Built in
Resolution	8 bit
Absolute accuracy (Including INL, DNL, quantization error, gain,offset error) [1]	2 LSB
Offset error (ADC, DAC) [1]	2 LSB
INL [1]	0.50 LSB
DNL [1]	0.25 LSB
Gain error [1]	2 LSB
Sampling frequency	9.6Hz
Operating temperature	-40°C ~ 85°C

Vi husker

Tabellen viser en 8 bits oppbygging med binære tallsystemet
Summen av tallene i øverste kolonne + 0 gir 256 muligheter.

128	64	32	16	8	4	2	1
1	1	1	1	1	1	1	1

[Link til konverterfunksjon](#)

`Serial.print("Button: "); buttonState og (" ")`
Angir at vi skal skrive Button, da det står i doble gåsetegn, så hente bryterens posisjon (høy eller lav) og så skrive dens posisjon i den serielle monitoren.

```
Button: 1 Pot: 0 Pot/4: 0  
Button: 1 Pot: 0 Pot/4: 0  
Button: 1 Pot: 0 Pot/4: 0  
Button: 1 Pot: 0 Pot/4: 0  
Button: 1 Pot: 0 Pot/4: 0
```

Det samme skal den gjøre med potensiometer verdien. Og på slutten ser vi den filtrerte verdien dvs. den som er dividert på 4.

```
Button: 1 Pot: 241 Pot/4: 60  
Button: 1 Pot: 241 Pot/4: 60  
Button: 1 Pot: 241 Pot/4: 60  
Button: 1 Pot: 241 Pot/4: 60  
Button: 1 Pot: 241 Pot/4: 60  
Button: 1 Pot: 241 Pot/4: 60  
Button: 1 Pot: 241 Pot/4: 60
```

Programmering Del 3

`Serial.println(filteredPotValue);` gjør at den printer neste data på en ny linje i stedet for at den printer på samme linje.

Du har nå vært gjennom mange nye elementer, og vi oppsummerer litt.

Sjekkpunkter, hva har jeg lært?

- _____ Jeg er kjent med Arduino UNO og IDE plattformen.
- _____ Jeg kan kople Arduino UNO til PC og laste ned programvare
- _____ Jeg kan finne frem til hvilke inn og utganger Arduino har
- _____ Jeg kan legge inn enkle program og forstå hvordan disse virker og forklare det
- _____ Jeg kan kople opp på Breadboard og teste programmet mitt
- _____ Jeg kjenner til noen av komponentene og deres funksjon

Beslutningstaking og bruk av logikk

I denne delen lærer du om:

- Bruke «if»-setninger
- Bruke «while»-løkker
- Bruke "for" løkker
- Bruke 'Switch cases'
- Bruke matematikk i Arduino
- Opprette funksjoner
- Opprette sofistikert kode

Ved slutten av denne delen, skal du kunne opprettet din egen kalkulator ved hjelp av ulike funksjoner, løkker, og Serielle funksjoner.

If setning

Først skal vi se på «if» uttrykket, som er grunnleggende i beslutningsforhold for mikrokontrollere. If uttrykket gir deg mulighet til å gå forbi en kode eller å gå inn i den avhengig av forholdet man har satt opp i koden. If funksjonen uttrykker hvis «dette» gjør «slik»

Gjennomgang av et oppsett hvor vi benytter «if» funksjonen til å bygge en vippe-bryter (toggle).

```
// pin definitions
int ledPin = 3;
int buttonPin = 2;
```

Man definerer inn og utganger, ledPin utgang på port 3 og buttonPin signal inn på port 2

```
// global variables
int toggleState;
int lastButtonState = 1;
long unsigned int lastPress;
int debounceTime = 20;
```

Programmering Del 3

Legger inn de globale variablene, int toggleState; som angir bryterens stilling, høy eller lav og int lastButtonState = 1; definerer den siste funksjon på bryteren, kan være 0 og 1. Long unsigned int lastPress; dobler datalagringskapasiteten i tid og minimerer muligheten for negative verdier- som normalt er 50% av signalene og 50% er positive. I praksis arbeider vi med tid, og denne funksjonen skal bidra til at man minimerer muligheten til å få negative verdier som ikke er ønskelig.

Int debounceTime = 20 ; er en funksjon som skal forhindre «kontakt prell» når man trykker på en bryter. Tenk deg at du trykker på en bryter og på grunn av en klokkefrekvens får man egentlig mange av og på- «funksjoner» uten at man er klar over dette. For å sikre at du får den riktige funksjonen (av eller på) legges det inn en tidsramme i variablene for nettopp å sikre at av eller på-stilling er riktige.

Legg merke til at oppgavene står inne i krølleparentes, som indikerer at det hører mer til hver av operasjonene.

```
void setup() {  
  // setup pin modes  
  pinMode(ledPin, OUTPUT);  
  //pinMode(LED_BUILTIN);  
  pinMode(buttonPin, INPUT_PULLUP);
```

Programmering Del 3

```
}
```

Void setup (), går en gang. Her setter vi pinMode funksjonene opp mot inn, utgangene vi listet opp i starten, henholdsvis port 3 som utgang til LED og port 2 som inngang fra bryter, og bruker pullup funksjonen som beskrevet tidligere.

```
void loop() {  
  int buttonState = digitalRead(buttonPin); //read the state of buttonPin and store it as buttonState (0 or 1)
```

Void loop(), int buttonState =digitalRead(buttonPin); Denne funksjonen leser av status på bryterens inngang (port3) og lagrer denne som høyt eller lavt signal (0 eller 1).

```
if((millis() - lastPress) > debounceTime) //if the time between the last buttonChange is greater than the  
debounceTime  
{  
  lastPress = millis(); //update lastPress  
  if(buttonState == 0 && lastButtonState == 1) //if button is pressed and was released last change  
  {
```

if((millis() - lastPress) > debounceTime) Hvis den innspilte tiden, minus siste trykk er større en den satte prell tiden, som vi har satt til å være null- dvs. ikke angitt noe tid- da settes den til 0).

Programmering Del 3

La oss forklare funksjonen `millis`, som har en annen funksjon en `delay`.

`Delay` funksjonen er enkel å bruke, men fører til at programmet går inn i en form for ventefunksjon til forsinkelsen er ferdig. `Millis` er en software timer.

Eksempel:

Det går 5 sekunder før vi trykker på bryteren, dvs. 5000msek. Minus 0, er større en 20ms i `debounceTime`.

Hvis vi benytter `millis` funksjonen for timing, registrerer vi tiden fra en handling finner sted og ut tids- beregningsperioden (i vårt tilfelle `debounceTime`), for å avgjøre om det er et reelt trykk eller en «prell». På denne måten registreres en tidslinje for å sikre at vi får riktig funksjon på bryteren, dvs. er den høy eller lav.

`lastPress = millis()`; denne funksjonen oppdaterer siste reelle trykk basert på `millis` utregning.

`if(buttonState == 0 && lastButtonState == 1)` Hvis bryterstatus er lik null(sjekkes) OG siste bryterstatus var 1(sjekkes) – dvs. begge må være sanne.

(En operand som kontrollerer dvs. dobbelt `==` tegn betyr sjekker om den er lik, vi ser mer på disse funksjonene i et senere kapittel.), dobbelt `&&` betyr OG(AND)- som en OG port.

Les mer om millis på denne siden: [«Using millis»](#)

Test gjerne ut eksemplene for å bli bedre kjent med millis funksjonen. I vårt tilfelle benytter vi oss av del 4 i eksemplene.

```
{
  toggleState =! toggleState;           //toggle the LED state
  digitalWrite(ledPin, toggleState);
  //digitalWrite(LED_BULTIN,toggleState);
  lastButtonState = 0; //record the lastButtonState
}
```

`toggleState =! toggleState;` Utropstegnet her betyr NOT, dvs. denne funksjonen har en inverterende funksjon. Den konkluderer at hvis `toggleState` er 0, så er den ikke =, men 1 og omvendt.

`digitalWrite(ledPin, toggleState);` Her sender den ut signal `toggleState` til `ledPin`- en vår, port 3 som får den til å lyse eller ikke lyse avhengig av `toggleState`.

`lastButtonState = 0;` Registrerer at bryteren er blitt trykket, så neste funksjon må bli en frigjøring og lagrer den.

Programmering Del 3

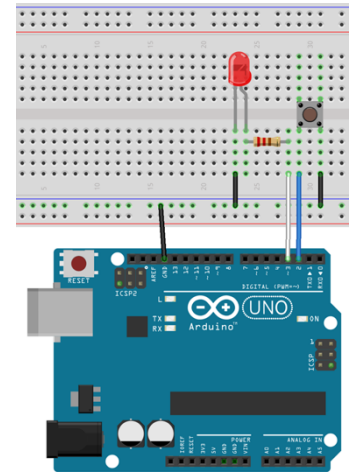
```
if(buttonState == 1 && lastButtonState == 0) //if button is not pressed, and was pressed last change
{
  lastButtonState = 1; //record the lastButtonState
}
}
```

if(buttonState == 1 && lastButtonState == 0) Hvis bryterstatus er lik en(sjekkes) OG siste bryterstatus var 0(sjekkes) – dvs. begge må være sanne.

Hvis bryterstatus er 1 er den ikke blitt trykket på, er den 0 er den nettopp blitt trykket på så ved å sette lastButtonState = 1; slik at vi resetter vårt program.

Kople opp en ekstern LED og bryter, legg inn kodene, last ned programmet og test funksjonen.

Ved hjelp av en enkel endring kan man benytte den innebyggede LED sammen med den eksterne, slik at man parallelt får en indikasjon på at funksjonen fungerer, indikator funksjon



Programmering Del 3

Hvorfor?

La oss tenke at denne funksjonen skal aktivere noe vi ikke ser, da kan man benytte slike oppsett for å gi indikasjon på at signalet er sendt ut og enheten vi ønsker å starte- har startet.

Hvordan?

[Link til kode](#)

While løkker

Nå skal vi se på «While» uttrykket, som er en viktig del i programmeringen. Denne funksjonen gir oss muligheter til å få en funksjon til å løpe kontenerlig- så lenge grunnlaget er tilstede. Den hopper ut av programmet når funksjonen ikke lenger er sann eller at man legger inn en stoppfunksjon i programmet.

I forhold til if løkkene løper denne funksjone om, om, og om igjen så lenge grunnlaget er tilstede.

Void loop funksjonen, som vi kjenner til, er egentlig bare en «fancy» while loop!

```
// pin definitions
int ledPin = 3;
int buttonPin = 2;

// global variables
int toggleState;
int buttonState = 1; //current state of the button, high

void setup() {
  // setup pin modes
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}
```

Programmering Del 3

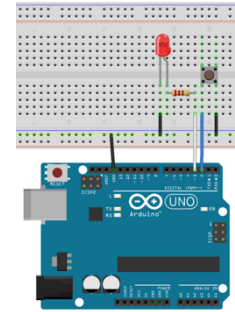
Denne innledningen begynner vi nå å kjenne, så kort fortalt, vi definerer inn og utganger, setter de globale variablene og forteller hvilke funksjoner som er koplet opp mot inn og utgangene vi har valgt (setup pin mode)

```
void loop() {  
  buttonState = digitalRead(buttonPin);  
  
  while(buttonState == 0)  
  {  
    toggleState = ! toggleState;  
    digitalWrite(ledPin, toggleState);  
    delay(50);  
    buttonState = digitalRead(buttonPin);  
  }  
  
  toggleState = ! toggleState;  
  digitalWrite(ledPin, toggleState);  
  delay(200);  
}
```

buttonState = digitalRead(buttonPin); buttonState er satt til høy som utgangspunkt (1). Inngangssignalet fra pinne 2, buttonPin leses (bryter inngang).

Programmering Del 3

`while(buttonState == 0)` Sammenligner variabelen til venstre(`buttonState`) med verdien eller variabelen til høyre for operatoren(`0`). Returnerer sann når de to operandene er like. Vær oppmerksom på at du kan sammenligne variabler av forskjellige datatyper, men det kan generere uforutsigbare resultater, det anbefales derfor å sammenligne variabler av samme datatype.



Verdien er satt til å være lik 0, dvs. lav (bryter ikke trykket inn). Når verdien ikke er sann mer, dvs. til vi trykker på bryteren, vil While funksjonen starte.

Funksjonen «`==0`» er en relasjons- operator eksempelvis som `X==Y`, som betyr X er lik Y, og i dette tilfellet betyr det at `buttonState`, dvs. bryterstatus er lik 0, som den vil være frem til noen trykker på den. Når noen trykker på bryteren, vil `toggleState = ! toggleState`; som er en bolsk operator (inverterende), invertere signalet og sende det ut på `digitalWrite (ledPin, toggleState)`; dvs. ut på `ledPin =3`;

EKSEMPEL:

`x != y`; er usann hvis x er lik y og det er sant hvis x ikke er lik y

- så vil den legge inn en pause på 50 millisekunder, `delay(50)`; før den igjen leser av `buttonState` for å få bekreftet bryterens status. Så lenge bryteren holdes nede, vil While

Programmering Del 3

funksjonen være aktiv og toggle, så derfor må den ha en tilbakemelding på bryterens status (`buttonState = digitalRead(buttonPin);`) for å overvåke om `while` funksjonen skal fortsette.

Når bryteren slippes, eller ikke er trykket inn, hopper den over `while` løkken, fordi den ikke lenger er sann, og inverterer signalet (toggler) mellom høy og lav, men da med 200 millisekunders pause.

??

Hva måtte du gjort for å endre funksjonen til å blinke raskt fra utgangspunktet og lengre mellomrom når du trykker på bryteren?

Hva måtte man tilføyet for å stoppe `while` løkken etter en gjennomgang

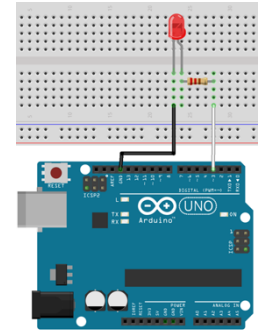
[Link til kode](#)

For løkker

I denne delen ser vi på «for» uttrykket, som brukes til å kontrollere flyten i programmet.

for -løkken brukes til å gjenta en løkke satt i klammeparenteser. En økningsteller brukes vanligvis til å øke og avslutte løkken.

for er nyttig for alle repeterende operasjoner, og brukes ofte i kombinasjon med matriser og når man ønsker å angi grunnlaget for at løkken skal fortsette og hva den skal gjøre når den er ferdig. En typisk for løkke benyttes når vi vet hvor mange ganger loopen skal gå og du ønsker å benytte en variabel for å loggføre hvor mange ganger den går (teller).



En matrise er en rektangulær tabell av tall. Tallene er skrevet i rekker og kolonner, og rundt tallene er det store firkantede parenteser. Et eksempel på en matrise kan være:

$$\begin{bmatrix} 1 & 2 & 7 \\ 7 & 2 & 2 \\ 8 & 8 & 1 \end{bmatrix}$$

Vi sier at denne matrisen har 3 rader, 3 kolonner og 9 komponenter.

Det finnes egne regneregler for matriser (kalt matriseoperasjoner), og ved hjelp av matriseregning kan vi løse store sett av likninger med mange ukjente.

Eksempel 1:

Hvis du for eksempel bruker en multiplikasjon i verdiøkning, genereres det en logaritmisk progresjon:

```
for (int x = 2; x < 100; x = x * 1.5) {  
    Println (x);
```

- Denne genererer tallverdiene: 2,3,4,6,9,13,19,28,42,63,94- men mindre en 100 og Siden vi benytter int, vil den kun vise heltall, ikke desimaler!

Eksempel 2:

Du ønsker at for løkken skal gå 5 ganger og utføre noe hver gang, så gir du loopen din en instruks for en slik løsning.

```
for (int i ; int < 5 , i++), den første er satt til «0», da vi ikke angir noe etter semikolonet, «i»  
er en tellerfunksjon.
```


Programmering Del 3

```
// pin definitions
int ledPin = 3; // analog writhe function

// global variables
int rampTime = 2;

void setup() {
  // setup pin modes
  pinMode(ledPin, OUTPUT);
  //initialise serial port
  Serial.begin(9600);
}
```

Også denne innledningen begynner vi med kjente elementer, med et unntak:

```
, int rampTime = 2;
```

Kort oppsummert definerer vi inn og utganger, setter de globale variablene og forteller hvilke funksjoner som er koplet opp mot inn og utgangene vi har valgt, i dette eksemplet har vi også med den innebyggede serielle funksjonen. `Serial.begin(9600)`; så vi kan følge endringene på skjermen.

`ledPin=3` utgangen har den innebyggede analoge skrivefunksjonen pulsbredde modulatoren til å regulere lysstyrken i vår LED lampe.

Programmering Del 3

`int rampTime = 2;` angir at stegene mellom hver skiftning, i dette tilfellet LED lampen sin gradvise økning og senkning av lysstyrken med 2 millisekunder mellom hvert trinn, denne kan modifieres etter eget ønske for å se den gradvise stigningen, normal må man opp i 10-15 millisekunder for at menneskets øye skal oppfatte skifte i trinnene.

```
void loop() {  
  // ramp LED brightness up to max  
  for(int i = 0; i<256; i++)  
  {  
    analogWrite(ledPin, i);          // for løkke 1  
    delay(rampTime);  
    Serial.println(i);  
  }  
  
  // ramp LED brightness down to 0  
  for(int i = 255; i>0; i--)  
  {  
    analogWrite(ledPin, i);          // for løkke 2  
    delay(rampTime);  
    Serial.println(i);  
  }  
}
```

Programmering Del 3

Her er det satt inn 2 stk. for løkker.

Den første løkken for (`int i = 0; i < 256; i++`) (increment) starter fra 0 og går opp til 255, dvs. mindre en 256. Som nevnt tidligere er den digitale utgangen 8 bit, dvs. 0 til 255 (256 muligheter). `i++` betyr at vi setter inn en økning på `i` med 1 hver gang, og skriver resultatet til vår serielle monitor.

Den andre for løkken for (`int i = 255; i > 0; i--`) (decrement) starter på 255 og arbeider seg ned til 0 med en senkning på 1 hver gang. `i--` betyr at vi setter inn en senkning med 1 hver gang, og skriver resultatet til vår serielle monitor.

Når du laster opp programmet til Arduinoen din så husk at TX og RX pinnene som benyttes til den serielle monitoren er de samme pinnene som benyttes for å laste opp programmet ditt. Derfor last opp programmet først og så åpne den serielle monitoren.

UART Rx / Tx, I²-C databuss og SPI databuss er ulike datakommunikasjonstyper som støttes av Arduino UNO.

Rx/Tx = Transmit and Recive

I²-C = Serial Inter-Integrated Circuit buss (seriell grensesnittprotokoll)

Programmering Del 3

Økningsoperatoren ++ legger til 1 i operanden, og reduksjonsoperatoren -- trekker 1 fra operanden.

Både øknings- og reduksjons-operatorene kan enten komme før (prefix) eller etter (postfix) operanden. For eksempel: Når et intervall eller en reduksjon brukes som en del av et uttrykk, er det en viktig forskjell i prefiks- og postfix-formen. Hvis du bruker prefiksform, vil økning eller reduksjon bli gjort før resten av uttrykket, og hvis du bruker postfix-skjema, vil økning eller reduksjon bli gjort etter at hele uttrykket er evaluert.

"C" i `cout` refererer til "tegn" og "out" betyr "utgang", derfor betyr `cout` "tegnutgang". `Cout` -objektet brukes sammen med innsetningsoperatoren `<<` for å vise en strøm av tegn.

Test prefix or postfix:

https://www.tutorialspoint.com/compile_cpp_online.php

C++ opplæring:

<https://www.programiz.com/cpp-programming>

```
x = x+1;  
  
is the same as  
  
x++;
```

And similarly -

```
x = x-1;  
  
is the same as  
  
x--;
```

```
#include <iostream>  
using namespace std;  
  
main() {  
    int a = 21;  
    int c ;  
  
    // Value of a will not be increased before assignment.  
    c = a++;  
    cout << "Line 1 - Value of a++ is :" << c << endl ;  
  
    // After expression value of a is increased  
    cout << "Line 2 - Value of a is :" << a << endl ;  
  
    // Value of a will be increased before assignment.  
    c = ++a;  
    cout << "Line 3 - Value of ++a is :" << c << endl ;  
    return 0;  
}
```

```
Line 1 - Value of a++ is :21  
Line 2 - Value of a is :22  
Line 3 - Value of ++a is :23
```

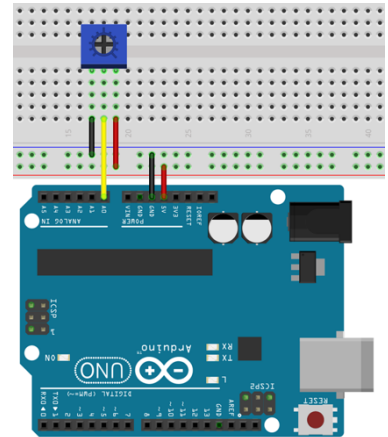
Switch case løkken

Nå skal vi ta en titt på «**switch case**» uttrykket, som lar deg bruke både variabel og verdi brukes til å sammenlikne lister av resultater. Tenk at du har en variabel som kan være et hvilket som helst tall mellom 0 og 9, og du ønsker at det skal være forskjellig utfall for hvert av tallverdiene. Da kan du bruke if løkker men i praksis vil man forenkle og heller bruke switch case funksjonen som gjør det mer oversiktlig og enklere og forstå.

```
// pin definitions
int potPin = A0;

// declare global variables
int lastPotValue;

void setup() {
  // set pin modes
  pinMode(potPin, INPUT);
  //initialise Serial port
  Serial.begin(9600);
}
```



Programmering Del 3

Vi angir hvilken pinne potensiometeret er koplet til int potPin = A0; og setter opp en global variabel int lastPotValue; som da leser potensiometeret sin posisjon.

I void setup () defineres først pinMode(potPin, INPUT); som inngang, og vi setter opp den interne serielle leseren med en «baud rate» på 9600, som er hastigheten på hvor fort den sender ut signalet.

Serial.begin() etablerer seriell kommunikasjon mellom Arduino-kortet og en annen enhet. Den vanligste bruken av seriell kommunikasjon du vil etablere er mellom Arduino og datamaskinen din via en USB-kabel - eller skal jeg si en Universal Seriell Bus-kabel. Når du har etablert seriell kommunikasjon mellom to enheter, tillater den at de to enhetene kommuniserer ved hjelp av en seriell protokoll.

Baude rate, Dette tallet(eks. 9600) kalles overføringshastigheten. Sannsynligvis er det viktigste at du forstår at tallet du setter i programmet og tallet som står i det serielle vinduet er like. Hvis du bruker Arduino IDE [Serial Monitor-vinduet](#) til å lese informasjonen fra Arduino, kan du bruke listen som beskriver overføringshastighet til å angi felles overføringshastighet.

MERK!!! :

Hvis disse to verdiene ikke samsvarer - vil alt du prøver å sende over seriell ikke fungere riktig.

KORT OPPSUMERT

1. `Serial.begin()` brukes til å etablere seriell kommunikasjon
2. Seriell kommunikasjon er en måte å tillate enheter å snakke med hverandre på
3. Ofte må du bruke `Serial.begin()` når du vil skrive ut noe på dataskjermen fra Arduino. Dette vil også kreve `Serial.println()`-funksjonen benyttes for å printe, sende.

[Link til kode](#)

Programmering Del 3

Under void loop () setter vi inn en kommando, denne leser potensiometerets verdi
`int potValue = analogRead(potPin) / 255;` Vi kjenner til fra tidligere at leseverdien analogt er på 10-bit, og utgangsverdien er på 8-bit, så vi legger inn en 8-bit-deling på 10 bitt analogt inngangssignal for å få 5 ønskede leseposisjoner.

`if (potValue != lastPotValue)` betyr at den leser `potValue` og sjekker om den er (ikke lik) ulik fra siste verdi lest inn `lastPotValue`. Dette kaller vi en relasjonsoperator! Og kontrollen gjøres for at programmet vårt skal følge med på stigningen etter som vi skrur på potensiometeret, opp eller ned (høyere eller lavere motstandsverdi). Det er disse analoge verdiene vi har delt inn i 5 ulike kategorier for å få 5 trinn i vårt oppsett case 0-4.

Oppsettet vårt er basert på at vi ønsker informasjon i forhold til hvilket nivå sensorene våre er på (i vårt tilfelle motstandsverdi) og hvordan vi da kan sende et utsignal basert på denne statusen.

Til slutt har vi satt inn en default status som forteller at hvis ingen av de oppsatte Case- forholdene er sanne, bruk default! Denne har vi lagt inn som en sikkerhetsramme, slik at vi får beskjed hvis noe feiler på brettet vårt, inngangssignalene eller i programmet vårt.

Legg merke til:

Vi bruker `Serial.begin()` når du vil skrive ut noe på dataskjermen fra Arduino. Dette vil også kreve `Serial.println()`-funksjonen som vi har satt inn i case- rekken.

Matematiske funksjoner

I denne delen ser vi på «**math**» uttrykket. Frem til nå har vi sett på en rekke andre forhold «lower level» funksjonene for å gjøre det «lettere» å forstå prosessene i oppstarten.

Mate funksjonen er ikke den ordinære matematikken vi kjenner til men binær matematikk og bit manipulasjon, som er mikrokontrollerens måte å behandle matematiske funksjoner på. Som vi vet er **int- Integer**, dvs. hele tall, måten mikrokontrolleren arbeider på, den liker ikke desimaler eller flytende tallverdier(fraksjoner), som vi også kaller for nettop **floatingPoint** nummre.

Floating point numbers tar også mer plass og mer kapasitet av mikroprosessoren, og med dettet også mer energi under drift.

Under denne seksjonen går vi kun innom basis funksjoner, og det finnes en rekke biblioteker for mer avanserte funksjoner og prosesser.

Vi kommer innom multiplikasjon, dividering, addisjon og subtraksjon. Symbolene vi benytter til disse operasjonene kaller vi for **opperander** slik som her vist.



Programmering Del 3

```
Int x = 10;
Int y = 3;
void setup() {
  Serial.begin(9600);
}
void loop() {
  int i = x+y ;
  serialPrintln(i);
  delay(1000);
}
```

Last ned og test på den serielle leseren.

Endre + til minus eller dele på osv. For å se funksjonen, husk å laste ned for hver gang og så åpne den serielle leseren.

Hvis du leger inn dele tegn må du også huske å endre int til float forran x+y faktoren

```
int i = x+y ;
```

```
float i = x/y ; // endres fra pluss til dele
```

Her vil du fremdeles få 3,00 som svar, da den lokale operanden er satt til int og ikke float

Programmering Del 3

```
int x = 10;
float y = 3; // endre også denne til float
void setup() {
  Serial.begin(9600);
}
void loop() {
  float i = x/y;
  serialPrintln(i);
  delay(1000);
}
```

Endrer den ene lokale operanden til float og vips... 3,33

En annen måte er å gjøre **i** om til **y**. Da blir den serielle utskriften 3,33 så 3,00 og så 3,33 osv. Hvorfor?

Fordi:

$y = x/y$ som gir 3,33 når den er satt til float

$3,33 = x/3,33$ blir da 3,00. $X = 3,00 \times 3,33$

Programmering Del 3

```
Int x = 10;
float y = 3; // endre også denne til float

void setup() {
  Serial.begin(9600);
}
void loop() {
  float y = x/y; // gjør i om til y
  serialPrintln(y); // husk også denne må da hete y og ikke i
  delay(1000);
}
```

Datatype for flyttall, et tall som har desimaltegn. Flyttall brukes ofte for å tilnærme seg analoge og kontinuerlige verdier fordi de har større oppløsning enn heltall. Flyttall kan være så store som 3,4028235E+38 og så lave som -3,4028235E+38. De lagres som 32 biter (4 byte) med informasjon.

11111111.11111111.11111111.11111111

Advarsel!

Hvis du utfører matematikk med «float», må du legge til et desimaltegn, ellers vil det bli behandlet som en int. Se denne siden for mer informasjon :

<https://www.arduino.cc/reference/en/language/variables/constants/floatingpointconstants/>

Programmering Del 3

Flytdatatype må maksimalt ha 6-7 desimaler for å være presis. Det betyr **totalt** antall sifre, ikke tallet til høyre for desimaltegnet. I motsetning til andre plattformer, hvor du kan få mer presisjon ved å bruke en dobbel (f.eks. opptil 15 sifre), på Arduino, er dobbel samme størrelse som flyt. Flyttall er derfor ikke nøyaktige, og kan gi merkelige resultater. For eksempel kan $6,0 / 3,0$ ikke være lik $2,0$. Se informasjon for utdyping.

<https://www.arduino.cc/reference/en/language/variables/data-types/float/>

Link til kode

Opprette egne funksjoner

I denne delen ser vi på «**opprette egne funksjoner**». Vi skal forsøke oss på modulære koder og gjenbrukbare koder. Eksempelvis en hypotenus kalkulator som vist i dette eksempelet.

```
void setup() {  
  //initialise Serial port  
  Serial.begin(9600);  
}  
  
void loop() {  
  int a;  
  int b;  
  float result;  
  
  //print instructions, and wait until there is something in the serial buffer  
  Serial.print("Enter a side value: ");  
  while(!Serial.available());  
  a = readSerial();  
  if(a == 0)  
  {  
    return;  
  }  
  Serial.print("Enter the other side value: ");
```

Programming Del 3

```
while(!Serial.available());
b = readSerial();
  if(b == 0)
  {
    return;
  }
findSide(a,b);

Serial.println();
}
```

//readSerial takes the next integer in the Serial buffer, clears the buffer, then returns it

```
int readSerial()
{
  int i = Serial.parseInt();

  //checks if the received value is a valid integer
  if(i < 1 || (i%1 != 0))
  {
    Serial.println("That isn't a valid integer");
    return 0;
  }
  Serial.println(i);
  Serial.parseInt();
  return i;
}
```

```
}  
  
void findSide(int x, int y)  
{  
  //calculate C squared by A squared + B squared  
  float hypotenuse = sqrt(x*x + y*y);  
  
  //print out the result  
  Serial.print("Hypotenuse = ");  
  Serial.println(hypotenuse);  
}
```

Funksjoner er viktige elementer i programmeringen og er en gruppe uttrykk som løper for å utføre en oppgave. Den kan være kort og enkel, som å lese verdier fra en sensor opp til svært kompleks. Funksjonen løper når den settes opp til dette og når den er ferdig returnerer den til hoved oppsett.

Arduino bruker 2 funksjoner i hvert oppsett(scetch), disse er setup end loop. Det er fire hovedområder i funksjonene:

- Function return type
- Function name
- Function parameters
- Function body

Disse funksjonene kan både hente data og sende data til programmet. Den første funksjonen er «return type» som er de data som returneres til programmet.

void er en retur type. Mer spesifikt returnerer ikke noe.

loop er en sekvens av instruksjoner som gjentas til en bestemt tilstand er nådd, eller de kjører uendelig i tilfelle den er satt opp med uendelige løkke.

I programmet vårt, som er Software basert, settes det opp en seriell port på 9600 baud rate.

```
void loop() {  
  int a;  
  int b;  
  float result;
```

Er lokale variabler. Leg merke til at vi bruker float i resultat for å få med desimaler.

Programmering Del 3

```
//print instructions, and wait until there is something in the serial buffer
Serial.print("Enter a side value: ");
while(!Serial.available());
a = readSerial();
  if(a == 0)
  {
    return;
  }
Serial.print("Enter the other side value: ");
while(!Serial.available());
b = readSerial();
  if(b == 0)
  {
    return;
  }
findSide(a,b);

Serial.println();
}
```

Serial.print("Enter a side value: "); Her ber den serielle skriveren vår deg om å taste inn data for a og b sidene. while(!Serial.available()); henter bits/bytes fra den serielle porten, dvs. data som du legger inn og lagres i den serielle mottaks buffer, som kan holde på inntil 64 bytes. Denne loopen går frem til begge får innkommende data (a og b).
a = readSerial(); denne funksjonen gjør at man leser innkommende serielle data.

Programmering Del 3

if(a == 0) sjekker om «a» er lik 0, hvis ja er den sann og intet skrives, hvis ikke er den ulik og skal skrive tallet return; (return value).

Litt informasjon om operatører.

Det første uttrykket er en **tilordningsoperator** "=", den andre er en **sammenligningsoperator** "==".

//readSerial takes the next integer in the Serial buffer, clears the buffer, then returns it

```
int readSerial()
{
  int i = Serial.parseInt();    //ser etter den neste godkjente int i den innkommende serielle informasjonen

  //checks if the received value is a valid integer
  if(i < 1 || (i%1 != 0))      //hvis i er mindre en 1(0 eller negativ)
  {
    Serial.println("That isn't a valid integer");
    return 0;                  //hvis verdiene er 0 eller negative returneres tallverdien 0 og teksten, ikke en
    godkjent verdi
  }
  Serial.println(i);
  Serial.parseInt();          //hvis verdien er lovlig, skriv ut verdien på seriell skriver
  return i;
}
```

Programmering Del 3

```
void findSide(int x, int y)
{
//calculate C squared by A squared + B squared
float hypotenuse = sqrt(x*x + y*y);

//print out the result
Serial.print("Hypotenuse = ");
Serial.println(hypotenuse);
}
```

`int readSerial()` henter neste integer (heltallet) i den serielle bufferen, tømmer bufferen, og returnerer tallet. `int i = Serial.parseInt();` ser etter den neste godkjente `int`(heltall) i den innkommende serielle informasjonen. `if(i < 1 || (i%1 != 0))` sjekker at `i(int)` er et heltall og større en 0

`Serial.println` skriver tekst om det ikke er riktig tall eller størrelse og viser tallet 0, `return 0;` hvis verdien er lovlig/riktig, skrives den ut på den serielle skriveren.

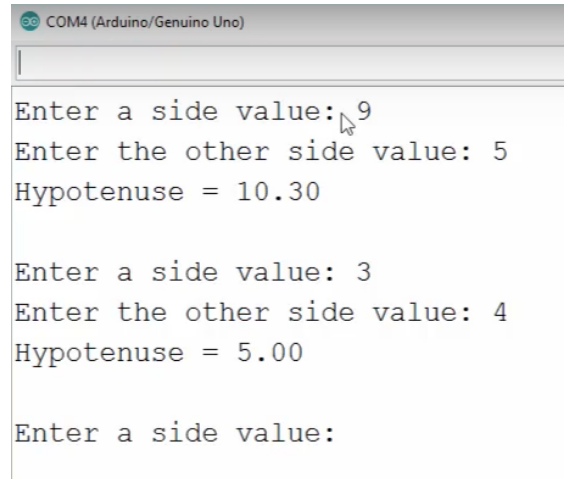
`Void findSide(int x, int y)` finner `int x` og `int y` basert på kontrollert og riktig innputt i den serielle monitoren.

`float hypotenuse = sqrt(x*x + y*y);` angir `float` for hypotenus svaret, slik at vi kan få ut desimale svar. Regner ut basert på kvadratroten av `x` og `y` data.

```
Serial.print("Hypotenuse = ");
Serial.println(hypotenuse);
```

Skriver ut teksten **Hypotenuse** og skriver data fra hypotenus til den serielle porten i ASCII format etterfulgt av en **Carriage return/ enter**

Eksempel på print kommando



```
COM4 (Arduino/Genuino Uno)
Enter a side value: 9
Enter the other side value: 5
Hypotenuse = 10.30

Enter a side value: 3
Enter the other side value: 4
Hypotenuse = 5.00

Enter a side value: 3
Hypotenuse = 5.00
```

- `Serial.print(78, BIN)` gives "1001110"
- `Serial.print(78, OCT)` gives "116"
- `Serial.print(78, DEC)` gives "78"
- `Serial.print(78, HEX)` gives "4E"
- `Serial.print(1.23456, 0)` gives "1"
- `Serial.print(1.23456, 2)` gives "1.23"
- `Serial.print(1.23456, 4)` gives "1.2346"

[Link til kode](#)

Sjekkpunkter, hva har jeg lært?

- _____ Jeg har lært hvordan man kan bruke if setninger
- _____ Jeg har lært hvordan man kan bruke while løkker
- _____ Jeg har lært hvordan man kan bruke for løkker
- _____ Jeg har lært hvordan man kan bruke Switch cases
- _____ Jeg har lært hvordan man kan bruke matematikk
- _____ Jeg har lært hvordan man kan opprette funksjoner
- _____ Jeg har lært hvordan man kan bruke sofistikerte koder og lage en hypotenus kalkulator

Datamanipulasjon og EEPROM

I dette kapitlet lærer du om:

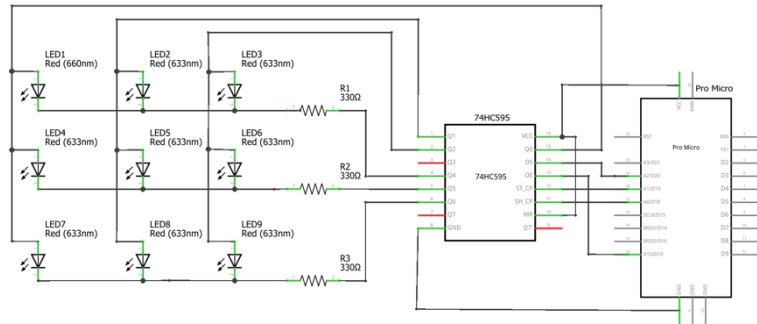
1. Bruke variabler i matriser
2. Bruke aritmetiske operatører, relasjonsoperatører, logiske operatører og tildelingsoperatører
3. Manipulere data ved hjelp av bit-vise operatører og logikk
4. Lagre informasjon mellom reseting ved hjelp av EEPROM



På slutten av dette kapitlet skal vi ha bygget opp en Boolsk LED funksjon ved hjelp av matriser, operatører og EEPROM- Lagring.

Innledningsvis vises en oversikt over:

- aritmetiske operatører
- relasjonsoperatører
- logiske operatører
- tildelingsoperatører
- bitvise operatører



Programming Del 3

Arithmetic operators [\[edit \]](#)

All arithmetic operators exists in C and C++ and can be overloaded in C++.

Operator name	Syntax	C++ prototype examples	
		As member of K	Outside class definitions
Addition	<code>a + b</code>	<code>R K::operator +(S b);</code>	<code>R operator +(K a, S b);</code>
Subtraction	<code>a - b</code>	<code>R K::operator -(S b);</code>	<code>R operator -(K a, S b);</code>
Unary plus (Integer promotion)	<code>+a</code>	<code>R K::operator +();</code>	<code>R operator +(K a);</code>
Unary minus (additive inverse)	<code>-a</code>	<code>R K::operator -();</code>	<code>R operator -(K a);</code>
Multiplication	<code>a * b</code>	<code>R K::operator *(S b);</code>	<code>R operator *(K a, S b);</code>
Division	<code>a / b</code>	<code>R K::operator /(S b);</code>	<code>R operator /(K a, S b);</code>
Modulo (integer remainder)^[a]	<code>a % b</code>	<code>R K::operator %(S b);</code>	<code>R operator %(K a, S b);</code>
Increment	Prefix	<code>R& K::operator ++();</code>	<code>R& operator ++(K& a);</code>
	Postfix	<code>R K::operator ++(int);</code>	<code>R operator ++(K& a, int);</code>
Note: C++ uses the unnamed dummy-parameter <code>int</code> to differentiate between prefix and postfix increment operators.			
Decrement	Prefix	<code>R& K::operator --();</code>	<code>R& operator --(K& a);</code>
	Postfix	<code>R K::operator --(int);</code>	<code>R operator --(K& a, int);</code>
Note: C++ uses the unnamed dummy-parameter <code>int</code> to differentiate between prefix and postfix decrement operators.			

Programming Del 3

Comparison operators/relational operators [\[edit \]](#)

All comparison operators can be overloaded in C++.

Operator name	Syntax	Included in C	Prototype examples	
			As member of K	Outside class definitions
Equal to	<code>a == b</code>	Yes	<code>bool K::operator==(S const& b) const;</code>	<code>bool operator==(K const& a, S const& b);</code>
Not equal to	<code>a != b</code> <code>a not_eq b</code> ^[b]	Yes	<code>bool K::operator!=(S const& b) const;</code>	<code>bool operator!=(K const& a, S const& b);</code>
Greater than	<code>a > b</code>	Yes	<code>bool K::operator>(S const& b) const;</code>	<code>bool operator>(K const& a, S const& b);</code>
Less than	<code>a < b</code>	Yes	<code>bool K::operator<(S const& b) const;</code>	<code>bool operator<(K const& a, S const& b);</code>
Greater than or equal to	<code>a >= b</code>	Yes	<code>bool K::operator>=(S const& b) const;</code>	<code>bool operator>=(K const& a, S const& b);</code>
Less than or equal to	<code>a <= b</code>	Yes	<code>bool K::operator<=(S const& b) const;</code>	<code>bool operator<=(K const& a, S const& b);</code>
Three-way comparison ^[c]	<code>a <=> b</code>	No	<code>auto K::operator<=>(const S &b);</code>	<code>auto operator<=>(const K &a, const S &b);</code>

Note: The operator has a total of 6 return types: `std::weak_equality`, `std::strong_equality`, `std::partial_ordering`, `std::weak_ordering`, `std::strong_ordering`, and `std::common_comparison_category`

Programming Del 3

Logical operators [\[edit \]](#)

All logical operators exist in C and C++ and can be overloaded in C++, albeit the overloading of the logical AND and logical OR is discouraged, because as overloaded operators they behave as ordinary function calls, which means that *both* of their operands are evaluated, so they lose their well-used and expected [short-circuit evaluation](#) property.^[1]

Operator name	Syntax	C++ prototype examples	
		As member of K	Outside class definitions
Logical negation (NOT)	<code>!a</code> <code>not a</code> ^[b]	<code>bool K::operator !();</code>	<code>bool operator !(K a);</code>
Logical AND	<code>a && b</code> <code>a and b</code> ^[b]	<code>bool K::operator &&(S b);</code>	<code>bool operator &&(K a, S b);</code>
Logical OR	<code>a b</code> <code>a or b</code> ^[b]	<code>bool K::operator (S b);</code>	<code>bool operator (K a, S b);</code>

Programming Del 3

Assignment operators [\[edit\]](#)

All assignment expressions exist in C and C++ and can be overloaded in C++.

For the given operators the semantic of the built-in combined assignment expression `a op= b` is equivalent to `a = a op b`, except that `a` is evaluated only once.

Operator name	Syntax	C++ prototype examples	
		As member of K	Outside class definitions
Direct assignment	<code>a = b</code>	<code>R& K: :operator =(S b);</code>	N/A
Addition assignment	<code>a += b</code>	<code>R& K: :operator +=(S b);</code>	<code>R& operator +=(K& a, S b);</code>
Subtraction assignment	<code>a -= b</code>	<code>R& K: :operator -=(S b);</code>	<code>R& operator -=(K& a, S b);</code>
Multiplication assignment	<code>a *= b</code>	<code>R& K: :operator *=(S b);</code>	<code>R& operator *=(K& a, S b);</code>
Division assignment	<code>a /= b</code>	<code>R& K: :operator /=(S b);</code>	<code>R& operator /=(K& a, S b);</code>
Modulo assignment	<code>a %= b</code>	<code>R& K: :operator %=(S b);</code>	<code>R& operator %=(K& a, S b);</code>
Bitwise AND assignment	<code>a &= b</code> <code>a and_eq b</code> ^[b]	<code>R& K: :operator &=(S b);</code>	<code>R& operator &=(K& a, S b);</code>
Bitwise OR assignment	<code>a = b</code> <code>a or_eq b</code> ^[b]	<code>R& K: :operator =(S b);</code>	<code>R& operator =(K& a, S b);</code>
Bitwise XOR assignment	<code>a ^= b</code> <code>a xor_eq b</code> ^[b]	<code>R& K: :operator ^=(S b);</code>	<code>R& operator ^=(K& a, S b);</code>
Bitwise left shift assignment	<code>a <<= b</code>	<code>R& K: :operator <<=(S b);</code>	<code>R& operator <<=(K& a, S b);</code>
Bitwise right shift assignment ^[e]	<code>a >>= b</code>	<code>R& K: :operator >>=(S b);</code>	<code>R& operator >>=(K& a, S b);</code>

Programmering Del 3

Bitwise operators [\[edit \]](#)

All bitwise operators exist in C and C++ and can be overloaded in C++.

Operator name	Syntax	Prototype examples	
		As member of K	Outside class definitions
Bitwise NOT	<code>~a</code> <code>compl a</code> ^[b]	R K: <code>operator ~();</code>	R <code>operator ~(K a);</code>
Bitwise AND	<code>a & b</code> <code>a bitand b</code> ^[b]	R K: <code>operator &(S b);</code>	R <code>operator &(K a, S b);</code>
Bitwise OR	<code>a b</code> <code>a bitor b</code> ^[b]	R K: <code>operator (S b);</code>	R <code>operator (K a, S b);</code>
Bitwise XOR	<code>a ^ b</code> <code>a xor b</code> ^[b]	R K: <code>operator ^(S b);</code>	R <code>operator ^(K a, S b);</code>
Bitwise left shift ^[d]	<code>a << b</code>	R K: <code>operator <<(S b);</code>	R <code>operator <<(K a, S b);</code>
Bitwise right shift ^{[d][e]}	<code>a >> b</code>	R K: <code>operator >>(S b);</code>	R <code>operator >>(K a, S b);</code>

[Link og referanse til operatorer i C++:](#)



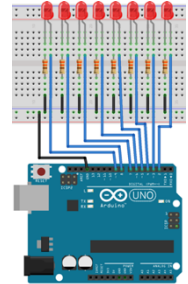
Rekker/matriser

Nå skal vi se på hvordan du bruker matriser til å lagre data, noe som gir mulighet for kraftigere variabel manipulerings.

Så langt har vi benyttet variabler i enkel utførelse. Nå skal vi se på en annen måte å bruke variabler på kaldt «[Arrays](#)»(Matriser). [Arrays](#) er en struktur som benyttes når flere data skal grupperes sammen. Tenk deg at Array er en kommode med mange skuffer, eller flere bokser på en hylle som inneholder hver sine interne variabler.

Det er 4 nøkkelementer i en [Array](#):

- Array data type (data type)
- Array unikt navn (unique name)
- Array indeks (Index)
- Array innhold (data type)



Programmering Del 3

Her har vi vist 4 ulike array oppsett:

- `Int arrayOne [4] = {11, 5, 4, 9};` ordinært oppsett
- `Int arrayTwo [] = {1, 2, 3, 4, 5, 6, 7, 8, 9};` ingen index
- `Int arrayThree [10];` angir senere
- `char arrayFour [6] = {"hello"};`
tekst, legg merke til index på 6 ikke 5 som hello er? Det skyldes utførelses tegnet/dataene som også er et tegn.

Første data begynner alltid med «0», 1 osv.

Du kan sette opp forskjellige typer av `array`, men en `array` kan kun inneholde en type data. Eksempelvis kan du starte med å deklare den som en `int`, da kan den kun inneholde `integer`. Som alle andre variabler kan `array` settes opp og deklarerer senere (legg til data senere). `Array` følger samme oppsett som andre variabler, den er unik, den er nøkkel sensitiv og har et globalt eller lokalt omfang.

Index refererer til `array` innholdet, når du setter opp en `array` kan du referere til antall elementer og plasser, hvis du ønsker. `Array` innholdet er de dataene du legger inn i din `array`.

Programmering Del 3

La oss se på et ferdig kodeoppsett som vi skal kople opp og teste!

```
int ledArray[] = {2,3,4,5,6,7,8,9}; ( våre utganger)
```

Her har vi definert `int` som data type, og da er det det vi må holde oss til. Den har fått et unikt navn, `ledArray`, Vi har ikke lagt noe inn i `arrays index`(den står tom `[]`) så vi har her ikke fortelt hvor mange elementer det er i vår `array`, men vi ser at det er 8 innenfor våre krølle parenteser(data).

```
int delayTime = 50;
```

Her er det lagt inn en `delay` mellom hver `int array`. Dvs. at den legger inn en pause på 50 millisekunder mellom hver utførelse.

```
void setup() {  
  //initialise ledArray as outputs  
  for(int i = 0; i<10; i++) // første gang denne kjører gir den i=0 som i vårt tilfelle er definert som 2 i  
  henhold til datainnhold for int.  
  {  
    pinMode(ledArray[i], OUTPUT); //ledArray[i] defineres som OUTPUT  
  }  
}
```


Programmering Del 3

Her settes `ledArray` opp som **OUTPUT** (utgang). Vi ser også at det benyttes en `for` løkke og setter opp `i` til å være `0` `i = 0`. `i` skal også være mindre enn `10`, `i = <10`, og at `i` teller oppover `i++` som er en aritmetisk operand. I vårt tilfelle er `i = 0`, som er `2` i vår datarekke `{2, 3, 4, 5, 6, 7, 8, 9}`;

```
void loop() {  
  //turn LEDs on from 0-7  
  for(int i = 0; i <= 7; i++)  
  {  
    digitalWrite(ledArray[i], HIGH);  
    delay(delayTime);  
  }  
  
  //turn LEDs off from 7-0  
  for(int i = 7; i >= 0; i--)  
  {  
    digitalWrite(ledArray[i], LOW);  
    delay(delayTime*5);  
  }  
}
```

Programmering Del 3

Her er det satt inn 2 stk. for løkker. Den ene sørger for å tenne LEDéne i stigende rekkefølge fra 0(utgang 2) til 7(utgang 9), og så fra 7 til 0 igjen.

`digitalWrite(ledArray[i], HIGH);` her sendes det digitale utgangssignalet høyt i henhold til `i`, som starter på 0 (`i=0`) og øker med en(`i++`) opp til 7(`i<=7`). Så her tenner en og en lysdiode ved hjelp av et høyt utgangssignal, `HIGH`.

`delay(delayTime);` her legges det inn en forsinkelse mellom hvert trinn fra 0 til 7.

Neste for løkke, `for(int i = 7; i >= 0; i--)` sendes det digitale utgangssignalet lavt i henhold til `i`, som starter på 7 (`i=7`) og minsker med en(`i--`) ned til 0(`i>=0`). Så her slår den av en og en lysdiode, ved hjelp av et lavt utgangssignal, `LOW`.

`delay(delayTime*5);` her legges det inn en forsinkelse på mellom hvert trinn fra 7 til 0 med en økning på *5 gangen.

Hva må endres/tilpasses for å få av og på linjert? dvs. fra 0 til 7 på og fra 0 til 7 av ...?

[Link til kode](#)

Operatører

I denne delen skal vi se på hvordan du bruker aritmetiske, relasjonelle, logiske og tildelingsoperatører. Disse brukes alle til å sammenligne og evaluere data. Operatørene, sybolene, forteller kompilatoren at den skal utføre en logisk eller matematisk oppgave.

Aritmetiske operatører har vi tidligere sett på når vi var innom matematikk og logiske funksjoner. Untaksvis har vi ikke sett på **Modulo**, **integer** remainder. Denne benyttes til å finne nettop, remainder, dvs. restverdi av et delbart tall.

Tenk deg at du legger inn følgende i Arduino IDE-kompilatoren.

Void setup() Setter først opp den serielle monitoren, til 9600.

Void loop(),

```
void setup() {  
  // put your setup code here, to run once:  
  Serial.begin(9600);  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
  int remainder = 11 % 3;  
  Serial.println(remainder);  
  delay(1000);  
}
```

Programmering Del 3

`Int remainder = 11 % 3`; her deler vi 11 på 3 (`11 % 3`;) og skriver ut restverdi til vår serielle printer. I dette tilfellet blir det 11/3, er 9 og `remainder` blir 2, dvs. restverdien etter full verdeling, som vi også ser på vår serielle monitor.

Endrer vi eksempelvis 11 til 15, `Int remainder = 15 % 3`; vil det komme 0 på vår serielle skriver, dvs. ingen restverdi fordi 15/3 er løsbart.

Arithmetic operators [edit]

All arithmetic operators exists in C and C++ and can be overloaded in C++.

Operator name	Syntax	C++ prototype examples	
		As member of K	Outside class definitions
Addition	<code>a + b</code>	<code>R K::operator +(S b);</code>	<code>R operator +(K a, S b);</code>
Subtraction	<code>a - b</code>	<code>R K::operator -(S b);</code>	<code>R operator -(K a, S b);</code>
Unary plus (integer promotion)	<code>+a</code>	<code>R K::operator +();</code>	<code>R operator +(K a);</code>
Unary minus (additive inverse)	<code>-a</code>	<code>R K::operator -();</code>	<code>R operator -(K a);</code>
Multiplication	<code>a * b</code>	<code>R K::operator *(S b);</code>	<code>R operator *(K a, S b);</code>
Division	<code>a / b</code>	<code>R K::operator /(S b);</code>	<code>R operator /(K a, S b);</code>
Modulo (integer remainder) ^[a]	<code>a % b</code>	<code>R K::operator %(S b);</code>	<code>R operator %(K a, S b);</code>
Increment	Prefix	<code>R& K::operator ++();</code>	<code>R& operator ++(K& a);</code>
	Postfix	<code>a++</code>	<code>R operator ++(K& a, int);</code>
<small>Note: C++ uses the unnamed dummy-parameter <code>int</code> to differentiate between prefix and postfix increment operators.</small>			
Decrement	Prefix	<code>R& K::operator --();</code>	<code>R& operator --(K& a);</code>
	Postfix	<code>a--</code>	<code>R operator --(K& a, int);</code>
<small>Note: C++ uses the unnamed dummy-parameter <code>int</code> to differentiate between prefix and postfix decrement operators.</small>			

Relasjonsoperatører

Relasjonsoperatører benyttes for å sjekke status på to verdier i stedet for å endre numrene. Som tidligere gjennomgått, husker vi det doble likhetstegnet for å bestemme om en verdi var lik eller ulike den andre.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}
```

```
x = 4
y = 6

(x == y) false
(x != y) true
(x > y) false
(x < y) true
(x >= y) false
(x <= y) true
```

Programmering Del 3

```
Void loop () {
Int x = 4;
Int y = 6;
```

```
If(x==y)
{
Serial.println("1");
}
```

```
else
{
Serial.println("0");
}
delay(1000);
}
```

Endre verdien til «not equal, greater then» og de andre relasjons operatorene, sjekk hva du får ut på den serielle monitoren, og skriv ned svaret!

Husk at! (utropstegn betyr not) ikke!

Comparison operators/relational operators [edit]

All comparison operators can be overloaded in C++.

Operator name	Syntax	Included in C	Prototype examples	
			As member of K	Outside class definitions
Equal to	a == b	Yes	<code>bool K::operator==(S const& b) const;</code>	<code>bool operator==(K const& a, S const& b);</code>
Not equal to	a != b a not_eq b <small>[R]</small>	Yes	<code>bool K::operator!=(S const& b) const;</code>	<code>bool operator!=(K const& a, S const& b);</code>
Greater than	a > b	Yes	<code>bool K::operator>(S const& b) const;</code>	<code>bool operator>(K const& a, S const& b);</code>
Less than	a < b	Yes	<code>bool K::operator<(S const& b) const;</code>	<code>bool operator<(K const& a, S const& b);</code>
Greater than or equal to	a >= b	Yes	<code>bool K::operator>=(S const& b) const;</code>	<code>bool operator>=(K const& a, S const& b);</code>
Less than or equal to	a <= b	Yes	<code>bool K::operator<=(S const& b) const;</code>	<code>bool operator<=(K const& a, S const& b);</code>
Three-way comparison ^[Q]	a <> b	No	<code>auto K::operator<>(const S &b);</code>	<code>auto operator<>(const K &a, const S &b);</code>

Note: The operator has a total of 6 return types: `std::weak_equality`, `std::strong_equality`, `std::partial_ordering`, `std::weak_ordering`, `std::strong_ordering`, and `std::common_comparison_category`.

Logiske operatører

Logiske operatører benyttes for å knytte flere relasjonsoperatører sammen. Vi har tidligere sett på et par av disse, `&&` og `!`. De logiske operatorene brukes ofte sammen med relasjonsoperatorene for å evaluere sannhetsverdi.

Eksempelvis: `((x < y) && (x == 0))`
Sjekk operanden i den serielle monitoren.

```
x = 0
y = 1

(x && y)  false
(x || y)  true
x = !x    x now equals 1 (not 0)
```

Kjenner vi igjen de boolske grunnportene/verdiene?

`!a` = not
`a && b` = and
`a ||` = or

```
if (! (x < y))
```

Sjekk operanden i den serielle monitoren.

NB! Ikke bland (`!a` som er ikke a) og (`a != b` som er ikke lik med b)

Eksempel, sjekk i den serielle monitoren: `if (! (x < y))`

Eksempel, sjekk i den serielle monitoren: `if ((x < y))`

Et lite tegn og vi endrer fra sann til usann, eller inverterer «sannheten»

Programmering Del 3

Tildelings operatorer

Tildelings operatorer er en snarvei i bruk av aritmetiske operatorer og bit-vised operatorer.

Eksempel:

```
Void loop () {
Int i = 3;
serial.println(i);
```

```
i = i + 5; (kan så forkortes til i += 5 også gjeldene for de øvrige aritmetiske operandene)
serial.println(i);
delay (1000);
}
```

Kjør sekvensen i den serielle monitoren. Skal da vise 3, 8, 3, 8 osv. Test den med den forkortede tildelings operanden og de aritmetiske operandene også.

Logical operators [\[edit \]](#)

All logical operators exist in C and C++ and can be overloaded in C++, albeit the overloading of the logical AND and logical OR is discouraged, because as overloaded operators they behave as ordinary function calls, which means that *both* of their operands are evaluated, so they lose their well-used and expected [short-circuit evaluation](#) property.^[1]

Operator name	Syntax	C++ prototype examples	
		As member of K	Outside class definitions
Logical negation (NOT)	<code>!a</code> <code>not a</code> ^[B]	<code>bool K::operator !();</code>	<code>bool operator !(K a);</code>
Logical AND	<code>a && b</code> <code>a and b</code> ^[B]	<code>bool K::operator &&(S b);</code>	<code>bool operator &&(K a, S b);</code>
Logical OR	<code>a b</code> <code>a or b</code> ^[B]	<code>bool K::operator (S b);</code>	<code>bool operator (K a, S b);</code>

```
i -= 2 same as i = i-2
i *= 2 same as i = i*2
i /= 2 same as i = i/2
```

BIT- MATEMATIKK

I denne delen skal du lære om å kontrollere enkeltbiter i byte-strenger ved hjelp av bit-vise operasjoner og logikk. Mange av variablene vi har benyttet til nå har vært basert på høy og lav(1 og 0) variabler, bollske uttrykk og integer variabler. Vi benytter da kun en bits data i en 16 bits rekke- som arduinoen kan håndtere. Det er svært verdifuldt å kunne kontrollere data på et lavere nivå og det er dette som kalles for Bit- matte og Bit-vise operander.

Vi skal gå gjennom ulike operander slik som and, or, not, xor og bit skifings operander. Ved hjelp av disse funksjonene forstå hvordan ulike bitsnummere kan bli manipulerte.

Leg merke til symbolene for funksjonene.

Eksempel: En 4 bits funksjon
og AND (&) operand:

Denne operanden trenger da 2 inn funksjoner for å kunne lage en ut funksjon, derav «og» funksjons navnet.

0 0 1 1

1 0 0 1

0 0 0 1

Vi ser at det er kun en funksjon som indikerer et signal ut og det er den med begge funksjoner høye, dvs. 1.

Bitwise operators [\[edit \]](#)

All bitwise operators exist in C and C++ and can be overloaded in C++.

Operator name	Syntax	Prototype examples	
		As member of K	Outside class definitions
Bitwise NOT	<code>~a</code> <code>compl a [b]</code>	<code>R K::operator ~();</code>	<code>R operator ~(K a);</code>
Bitwise AND	<code>a & b</code> <code>a bitand b [b]</code>	<code>R K::operator &(S b);</code>	<code>R operator &(K a, S b);</code>
Bitwise OR	<code>a b</code> <code>a bitor b [b]</code>	<code>R K::operator (S b);</code>	<code>R operator (K a, S b);</code>
Bitwise XOR	<code>a ^ b</code> <code>a xor b [b]</code>	<code>R K::operator ^(S b);</code>	<code>R operator ^(K a, S b);</code>
Bitwise left shift ^[d]	<code>a << b</code>	<code>R K::operator <<(S b);</code>	<code>R operator <<(K a, S b);</code>
Bitwise right shift ^[d]	<code>a >> b</code>	<code>R K::operator >>(S b);</code>	<code>R operator >>(K a, S b);</code>

Programmering Del 3

```
//AND
  x = 0011 0100
  y = 0101 1101
x & y = 0001 0100 // vi ser I denne 8 bits rekken får vi høy i felt 3 og 5 sett fra høyre side.
```

Eksempel: En 4 bits funksjon og OR (|) operand:

Denne operanden trenger da 1 inn funksjon for å kunne lage en ut funksjon, derav «eller» funksjons navnet.

```
0 0 1 1
1 0 0 1
1 0 1 1
```

Vi ser at det er her tre funksjoner som indikerer et signal ut og det er dem hvor en av funksjonen er høye, dvs. 1.

```
//OR
  x = 0011 0100
  y = 0101 1101
x | y = 0111 1101 // vi ser I denne 8 bits rekken får vi høy i felt 1, 3, 4, 5, 6 og 7 sett fra høyre side.
```

Programmering Del 3

Eksempel: En 4 bits funksjon og XOR (^) operand (Eksklusive OR funksjon):

Denne operanden trenger da kun en høy inn funksjon(eksklusive) for å kunne lage en ut funksjon, derav «eksklusive eller funksjon».

```
0 0 1 1  
1 0 0 1  
1 0 1 0
```

Vi ser at det er her to funksjoner som indikerer et signal ut og det er dem hvor kun en av funksjonen er høye, dvs. 1.

```
//XOR  
x = 0011 0100  
y = 0101 1101  
x ^ y = 0110 1001 // vi ser I denne 8 bits rekken får vi høy i felt 1, 4, 6 og 7 sett fra høyre side.
```

Programmering Del 3

Eksempel: En 4 bits funksjon og NOT (\sim) operand (Invertert funksjon):

Denne operanden inverterer signalet, en høy blir lav og en lav blir høy.

0 0 1 1 blir 1 1 0 0

1 0 0 1 blir 0 1 1 0

Vi ser at hver input inverteres og endrer ut signalene.

//NOT

x = 0011 0100

\sim x = 1100 1011 Invertert x (not)

y = 0101 1101

\sim y = 1010 0010 Invertert y (not)

Programmering Del 3

Eksempel: En 4 bits funksjon og SHIFT LEFT/RIGHT (<< / >>)operand (angis med skifte nummer):

Denne operanden skifter plasser i henhold til skifte nummer.

<<² **0 0 1 1 blir 1 1 0 0** **Skifter 2 plasser mot venstre**
<<¹ **0 0 1 1 blir 0 1 1 0** **Skifter 1 plass mot venstre**

>>² **0 0 1 1 blir 0 0 0 0** **Skifter 2 plasser mot høyre**
>>¹ **0 0 1 1 blir 0 0 1 0** **Skifter en plass mot høyre**

```
//SHIFT LEFT
x = 0011 0100
x << 2 = 1101 0000 // skifter 2 plasser left
y = 0101 1101
y << 2 = 0111 0100 // skifter 2 plasser left
```

Programmering Del 3

Vi skal nå se på en funksjon som bruker **shift** registre som gjør det mulig å utvide innganger og utganger på din mikrokontroller, ved hjelp av noen få kontrollenheter.

For å kontrollere **shift** registrene trenger vi klokke tilkopling, data tilkopling og en oppdaterings tilkopling. Klokke tilkoplingen hentes fra Arduinoen, som tillater at en og en bitt sendes ut av gangen i en seriell struktur på 8 bit-sekvens. Disse dataene mottas av **shift** registeret serielt og sendes til utgangene i en parallell struktur. Den enheten vi skal benytte i denne øvelsen har 8 kanaler, som vi sender data til. Det finnes både, **parallele innputt/serielle output** som brukes til å legge til ekstra inngangssignaler og motsatt, for å legge til flere utgangssignaler. De fungerer ikke sammen så i dette tilfellet skal vi arbeide med **output** funksjonen først.

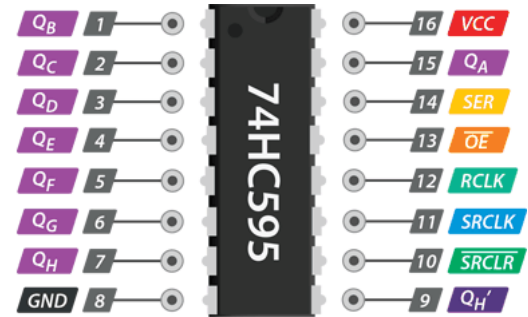
I øvelsen benytter vi en IC-krets (**shift** register) med en inngang, seriell og 8 utganger (74HC595). Enheten har også muligheter til å Daisy kople, dvs. samkjøre flere enheter. Vi benytter 8 LED i den parallele output funksjonen.

[Link til Data ark for 74HC595, 8-Bit Serial-Input/Serial or Parallel-Output Shift Register with Latched 3-State Outputs](#)

[Link til andre IC kretser, data register](#)

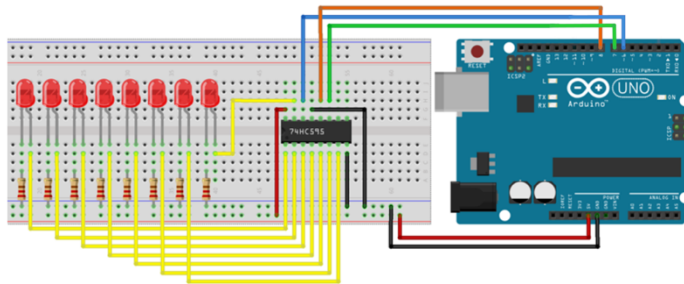
SIPO vs PISO skift registre

Skiftregistre kommer i to grunnleggende typer, enten SIPO (Serial-In-Parallel-Out) eller PISO (Parallel-In-Serial-Out). Den populære SIPO-brikken er **74HC595**, og PISO-brikken er 74HC165. Den første typen, SIPO, er nyttig for å kontrollere et stort antall utganger, som lysdioder og andre aktivatorer. Mens sistnevnte type, PISO, er bra for å samle et stort antall innganger, som brytere; som den som brukes i Original Nintendo Controller.



74HC595 Pinout

Koblings skjema for 'AND, OR, XOR kalkulator ved hjelp av SIPO



Koplingspinner 74HC595

GND skal kobles til GND av Arduino.

VCC er strømforsyningen for 74HC595 som vi kobler til 5V-pinnen på Arduino.

SER (Serial Input)-pinnen brukes til å mate data inn i skiftregisteret litt om gangen.

SRCLK (Shift Register Clock) er klokken for skiftregisteret. 595 er klokke-drevet høy. Dette betyr at for å skifte biter inn i skiftregisteret, må klokken være HØY. Og biter overføres inn på den høy delen av klokken.

RCLK (Register Clock / Latch) er en veldig viktig pin. Når denne blir kjørt HIGH, kopieres innholdet i Shift Register til «storage / latch» register; som til da dukker opp på utgangen. Så latch utgangen kan sees på som det siste trinnet i prosessen for å lede resultatene våre ut på utgangen, som i dette tilfellet er styring av lysdioder.

SRCLR (Shift Register Clear) pin lar oss resette hele Shift Register, noe som gjør alle bitene 0, samtidig. Dette er en negativ logikkpinne, så for å utføre denne tilbakestillingen; vi må sette SRCLR pin LOW. Når ingen tilbakestilling er nødvendig, skal denne utgangen være HØY.

Programmering Del 3

OE (Output Enable) er også negativ logikk: Når spenningen på den er HØY, er utgangspinnene deaktivert / satt til høyimpedans-tilstand og tillater ingen flyt av strøm. Når OE får lav spenning, fungerer utgangspinnene normalt.

QA-QH (Output Enable) er utgangspinnene kobles til en eller annen type signalbehandler eksempelvis lysdioder, 7 segmenter, releer etc.

QH'S Pin sender bit 7 av ShiftRegister. Det er slik at vi kan daisychain (kople flere enheter sammen, eksempelvis 2 stk 595 for å drive 16 LED). Hvis du kobler denne QH' til SER-pinnen til en annen 595, og gir begge ICene det samme klokkesignalet, vil de oppføre seg som en enhet med 16 utganger. Selvfølgelig er denne teknikken ikke begrenset til to ICs - du kan kople sammen så mange du vil, hvis du har nok strøm til dem alle.

```
const int dataPin = 6;           // Kobles til data pin på 74Hc- SER
const int clockPin = 7;          // Kobles til Clok på 74Hc- SRCLK
const int latchPin = 8;         // Kobles til Latch pin på 74Hc- RCLK

byte ledMap = 0b11111111;       // 0b betyr byte, 0x betyr hex desimal. 11111111 er oppløsningen (FF
// HEX)
int delayTime = 3000;
```


Programmering Del 3

Vi starter med å definere hva som er forskjellen på `int` og `const int`. `Const` står for konstant og betyr at den har en «`read only`» funksjon, `int` er en lese og skrive funksjon. Fordelen med denne er at den får en klarere funksjon tildelt og at det brukes mindre minne (1 byte i stede for 2).

`Byte ledMap = 0b11111111`; settes opp som en singel byte variabel slik at den ikke tar 2 byte slik som `int`. `LedMap` er en `map` funksjon som tilordner et tall fra ett område til et annet. Når vi angir den slik som her `1111 1111` vil arbeidsområdet ligge mellom 0-255- det må uansett være 8 bit/1byte.

`Int delayTime = 3000` er tidsforsinkelsen

```
void setup() {  
  // put your setup code here, to run once:  
  pinMode(dataPin, OUTPUT);  
  pinMode(clockPin, OUTPUT);  
  pinMode(latchPin, OUTPUT);  
  
  Serial.begin(9600);  
  shiftWrite(0x00);  
  Serial.println("Enter a number between 0-255");  
}
```

Programmering Del 3

Alle utganger må tildeles funksjon. Her angir vi alle 3 til å være digitale OUTPUT funksjoner, både `dataPin`, `clockPin` og `latchPin`. `LatchPin` er oppdaterings pinne som initierer utgangssignalene.

Setter opp vår serielle monitor `Serial.begin(9600);`

`Shift write(0x00);` er en hex kode(indikeres med `0x`) som setter alle lysdiodene våre til 0(lav).

`Serial.println("Enter a number between 0-255");` er en instruksjon som kommer opp på den serielle monitoren vår og ber deg sette inn et tall i henhold til `0b11111111` som vi initierte under `byte ledMap=0b11111111;`

Seriell kode sendes ut med 10 10 10 10 og skifter til venstre 01 01 01 01 01

Rekken vil da se noenlunde slik ut: 10101010-01010101-10101010 3 byte 8 bits rekker
(NB! Bindestrøk er ikke en del av rekken)

Programmering Del 3

```
void loop() {  
  if(Serial.available())  
  {  
    int inputVal = Serial.parseInt();  
  
    if(inputVal > 255)  
    {  
      Serial.println("Uh oh, try again");  
      Serial.println("Enter a number between 0-255");  
      return;  
    }  
  }  
}
```

Innledningsvis sjekker den om det ligger data i vår serielle monitor `if(Serial.available())`, den benytter denne tallverdien `int inputVal=Serial.parseInt()`; og sjekker om verdien er i henhold til angitte verdi oppgit i `ledMap`, som for vår del er satt lavere en 255 `if(inputVal >255`.

Hvis den er feil, dvs. utenfor angitte ramme, skriver den en tekst som sier `Serial.println("Uh oh, try again");` eller prøv igjen.

Og på ny får du mulighet til å legge inn et nummer mellom `>255`, `Serial.println("Enter a number between 0-255");` .

Programming Del 3

```
Serial.print("DECIMAL: ");
Serial.println(inputVal);
Serial.print("BINARY: ");
Serial.println(inputVal, BIN);
Serial.println();

Serial.print("AND result: ");
Serial.println(ledMap & inputVal, BIN);
shiftWrite(ledMap & inputVal);
delay(delayTime);

Serial.print("OR result: ");
Serial.println(ledMap | inputVal, BIN);
shiftWrite(ledMap | inputVal);
delay(delayTime);

Serial.print("XOR result: ");
Serial.println(ledMap ^ inputVal, BIN);
shiftWrite(ledMap ^ inputVal);
delay(delayTime);
Serial.println();
Serial.println("Enter a number between 0-255");
```

Programmering Del 3

Skriverfunksjonen er satt opp slik at den skriver til den serielle monitoren vår. Starter med å skrive inn **DECIMAL**, for så å hente Input verdien vi legger inn, denne må da være innenfor vår angitte ramme >255 , `Serial.println(inputVal;)`. Og skriver denne **DECIMAL** verdien. Så gjør den det samme med **BINARY**, og skriver den binære verdien siden vi har angitt `Serial.println(inputVal, BIN;)` den til å være binær, og skriver den ut på den serielle monitoren.

For AND, OR og XOR gjør den det same, men i tillegg utføres den serielle kontrollen mellom verdien vi har lagt i MAP funksjonen og den vi skriver inn på den serielle monitoren i binær form, slik som i eksemplene vist tidligere:

0 0 1 1	Verdi i MAP
1 0 0 1	Verdi lagt i monitor
<hr/>	
0 0 0 1	Output resultat

Legg også merke til de bit vise operandene:

```
Serial.println(ledMap & inputVal, BIN); // & FUNKSJON
```

```
Serial.println(ledMap | inputVal, BIN); // OR FUNKSJON
```

```
Serial.println(ledMap ^ inputVal, BIN); // XOR FUNKSJON
```

Henter verdien i ledMap

= 1111 1111 verdi lagt i Map

Henter inputVal(input verdien) eks. 10

= 0000 1010 verdi lagt inn i seriell monitor

= **0000 1010 resultat verdi i & funksjon (AND)**

Delay(delayTime); henter forsinkelsen som er satt til 3000ms innledningsvis

Til slutt ber den deg om å legge inn et nytt tall i den serielle **monitoren** `serial.println("Enter a number between 0-255");`

```
void shiftWrite(byte value)
{
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, value);
  digitalWrite(latchPin, HIGH);
}
```

Funksjonen satt opp her drar latch utgangen LOW, dvs.forteller den at den må være klar til å motta data.

ShiftOut, setter opp klokke utgang, mest eller minst signifikante data først(rekkefølgen) og verdien som ligger klar for sending.

LatchPin settes høy og data kopieres fra **shift** registeret til **storage/latch** og dukker opp på vår/våre utganger, eksempelvis LED eller også på monitoren vår.

RCLK (Register Clock / Latch) Når denne blir kjørt HØY, kopieres innholdet i Shift Register til «storage / latch» register; som til da dukker opp på utgangen våre. Så latch utgangen kan sees på som det siste trinnet i prosessen for å lede resultatene våre ut på utgangen, som i dette tilfellet er styring av lysdioder, når den står LAV venter den på nye data i shift registeret den kan sende ut når pulsen igjen blir HØY.

I informatikk er en bit den minste meningsfulle informasjonene vi kan bruke. Det uttrykkes oftest som et siffer i det binære tallsystemet: enten 0 eller 1. En streng på 8 biter kalles en byte. Hvis vi for eksempel tar det binære tallet 11100111 (231 i desimalverdi), og sender det som en datastreng i et nettverk, kan vi sende det på to måter: starte fra venstre til høyre, eller starte fra høyre til venstre. Disse to forholdene kalles Most Significant Bit First og Least Significant Bit First.

[Link til kode](#)

EEPROM

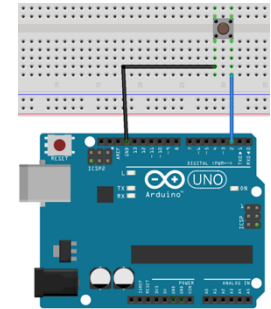
I denne delen lærer vi om lagring av data i det ikke-flyktige minnet kjent som **EEPROM**. Dette gjør at data kan opprettholdes når strømmen kobles fra og få tilgang til dataene senere.

Den fungerer på en måte som en liten Harddisk integrert inne i Chipen vår. I Arduino Uno Atmega 328/328p std. oppsett er det 1024(1K) i lagringskapasitet. Dette kan utvides ved å kjøpe tilsvarende enheter med større **EEPROM** eller andre tilleggsenheter. Ved hjelp av det innebyggede **EEPROM** biblioteket i Arduinoen kan du relativt enkelt få tilgang.

Husk at **EEPROM** har begrensede levetid i henhold til å skrive til den, før den begynner å tulle/feile. For Arduino UNO Atmega 328 er dette ca. 100.000 ganger. For å holde oversikt over antall ganger man skriver til en adresse, brukes «**Wear Leveling**». Ved hjelp av dette kan man endre adressen når den nærmer seg maks antall i telleren, eksempelvis 100.000.

I denne øvelsen skal vi benytte oss av et innebygget bibliotek i Arduinoen. Oppkoplingen er svært enkel og gjøres ved hjelp av en bryter som brukes som en teller.

Innebyggede biblioteker gjør det enklere å bruke og inneholder ofte mange ulike innebyggede funksjoner. Som programmerer trenger man ikke alltid å finne opp kruttet på nytt.



Programmering Del 3

```
#include <EEPROM.h>
```

`#include` brukes til å inkludere eksterne biblioteker `<EEPROM.h>` i skissen din. Dette gir programmereren tilgang til en stor gruppe standard C-biblioteker (grupper av forhåndslagde funksjoner), og også biblioteker skrevet spesielt for Arduino.

```
int ledPin = 13;  
int buttonPin = 2;
```

```
// global variables  
int lastButtonState = 1;  
long unsigned int lastPress;  
int debounceTime = 20;  
int counter;
```

Mye av dette oppsettet kjenner du muligens igjen fra toggle funksjonen vi arbeidet med tidligere.

`int lastButtonState = 1;` vi setter høyt utgangssignal
`long unsigned int lastPress;` som gir rom for lagring av mer datamengde
`int debounceTime = 20;` som forhindrer prell og sikrer et klart signal(høy/lav)
`int counter;` angir EEPROM chip som du ønsker å lese fra.

Programmering Del 3

```
void setup() {  
  // setup pin modes  
  pinMode(ledPin, OUTPUT);  
  pinMode(buttonPin, INPUT_PULLUP);  
  
  //initialise Serial port  
  Serial.begin(9600);
```

```
int buttonState = 0; // current state of the button  
int lastButtonState = 0; // previous state of the button  
int ledState = 0; // remember current led state
```

Setter opp pin mode, `ledPin` som `OUTPUT` og `buttonPin` som `INPUT_PULLUP`. Pullup funksjonen gikk vi gjennom under `if` løkkene, som tvinger den følsomme inngangen til å holde et fast signal.

`Serial.begin(9600);` Setter opp den serielle porten/monitoren.

```
//assign counter the value of address 0  
counter = EEPROM.read(0);  
//write a 0 to address 0. This allows for consecutive resets to reset the counter  
EEPROM.write(0,0);  
}
```

Programmering Del 3

`counter = EEPROM.read(0);` angir hvilken adresse den skal hente/lese data fra, som i dette tilfellet er (0). Leser da de siste dataene som ble lagt inn der, slik at de er tilgjengelig selv etter vi har slått av strømmen.

`EEPROM.write(0,0);` her skriver vi til adresse 0 og angir 0. Denne funksjonen legges inn slik at vi, på et senere tidspunkt, kan resette den lagrede verdien til 0, skrive 0 til adresse 0 ved å presse resett 2 ganger.

```
void loop() {  
  int buttonState = digitalRead(buttonPin); //read the state of buttonPin and store it as buttonState (0 or 1)  
  
  if((millis() - lastPress) > debounceTime) //if the time between the last buttonChange is greater than the  
  debounceTime  
  {  
    lastPress = millis(); //update lastPress  
    if(buttonState == 0 && lastButtonState == 1) //if button is pressed and was released last change  
    {
```

`int buttonState = digitalRead(buttonPin);` Leser inngangssignalet fra bryterinngang
`if((millis() - lastPress) > debounceTime,` kontrollerer at tiden mellom bryter trykk er større en `debounceTimen,` dvs. 20ms for at det skal være en gyldig registrering, og `millis` er telleren som teller opp mellom hvert trykk.

`lastPress = millis();` denne funksjonen oppdaterer siste reelle trykk

Programmering Del 3

`if(buttonState == 0 && lastButtonState == 1)` Hvis bryterstatus er lik null(sjekkes) OG siste bryterstatus var 1(sjekkes) – dvs. begge må være sanne.

(En operand som kontrollerer dvs. dobbelt `==` tegn betyr sjekker om den er lik, vi ser mer på disse funksjonene i et senere kapittel.), dobbelt `&&` betyr OG(AND)- som en OG port.

```
counter++;
EEPROM.write(0, counter); //write counter to address 0
digitalWrite(ledPin, HIGH); //momentary LED
lastButtonState = 0; //record the lastButtonState

//print the results
Serial.print("Counter: ");
Serial.println(counter);
}
if(buttonState == 1 && lastButtonState == 0) //if button is not pressed, and was pressed last change
{
  lastButtonState = 1; //record the lastButtonState
  digitalWrite(ledPin, LOW); //momentary LED
}
}
}
```

Programmering Del 3

Counter ++; teller hvert registrerbare trykk og skriver dette til minne EEPROM.Write(0, counter); .

Angir at denne kommandoen utføres og sender høyt signal ut på angitte LED utgang 13 som også er den interne LED, eller vi kan koble den opp selv på brettet.

Den registrer så siste bryterstatus som 0.

Vi skriver teksten «Counter» til den serielle monitoren og etter denne skrives verdien av tellerens siste oppdaterte sum, 1, 2, 3 osv.

`lastButtonState = 0;` Registrerer at bryteren er blitt trykket, så neste funksjon må bli en frigjøring og lagrer den.

`if(buttonState == 1 && lastButtonState == 0)` Hvis bryterstatus er lik en(sjekkes) OG siste bryterstatus var 0(sjekkes) – dvs. begge må være sanne.

Hvis bryterstatus er 1 er den ikke blitt trykket på, er den 0 er den nettopp blitt trykket på så ved å sette

`lastButtonState = 1;` slik at vi resetter vår program

[Link til kode](#)

Bibliotek, serielle data og forskjellig tillegg

- Bruke og inkludere biblioteker i prosjektene dine
- Bruke SPI-grensesnittet
- til å sende/motta serielle data
- Bruke I²-C-grensesnittet til å sende/motta serielle data
- Utvide prosjektet ditt med Arduino-maskinvareformatet kalt 'Shields'

SPI (Serial Peripheral Interface) er en høy hastighet, 3-wire, seriell kommunikasjonsprotokoll.

IC Serial Communication Protocol (også kalt I²C - Inter-Integrated Circuits) bruker to toveis kommunikasjonslinjer kalt Serial Data-bus (SDA) og SCL (Serial Clock) -bussen for å overføre data. Det er også to kraftledninger. SDA- og SCL-busser trekkes til kraftbussen gjennom motstander.

Shields er brett som kan plugges på toppen av Arduinoen din og utvider mulighetene. De forskjellige skjoldene følger samme filosofi som det originale verktøysettet, de er enkle å montere og billige å produsere.

SPI Interface

SPI Interface brukes som kommunikasjonsprotokoll for å sende og motta serielle data fra sensorer og liknende enheter. SPI er synkronisert datakommunikasjon og bruker klokkefrekvens for å regulere overføring av data. Den har også full Duplex protokoll som betyr at data kan sendes og mottas samtidig.

Clock

MISO, mastre inn, slave out

MOSI, master out, slave inn

Slave select line

SPI bruker ikke adressering og Slave Select linjen benyttes for å kommunisere med slavene. Master enheten genererer klokkefrekvensen. Data sendes og mottas i synkronisering med klokkepulsene. SPI kan operere på 4 ulike måter:

[Link til dypere SPI gjennomgang og funksjon](#)

```
#include <SPI.h>
```

Legger inn SPI bibliotek

```
int slaveSelect = 2;
```

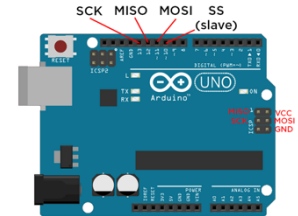
SPI har faste pinner på Arduino UNO:

```
13 = Klokke
12 = MISO
11 = MOSI
10 = SLAVE
```

Vi benytter tilkopling 2 som slaveSelect for å motta digitalt signal høyt eller lavt for å angi om en overførsel skal skje. Her kan du benytte enhver digital tilkopling.

```
int delayTime = 50;
```

Kontrollerer hastigheten, og i dette eksempelet den binære telleren opp og ned.



Programmering Del 3

```
void setup() {  
  pinMode(slaveSelect, OUTPUT);  
  SPI.begin();  
  SPI.setBitOrder(LSBFIRST);  
}
```

PinMode, slaveSelect som utgang. SPI.begin() initierer porten og SPI.setBitOrder(LSBFIRST), sender minst signifikante bitt først (kan endres, den angir bare rekkefølgen for LEDene)

```
void loop() {  
  for (int i; i < 256; i++) //For loop to set data = 0 then increase it by one for every iteration of the loop,  
    when the counter reaches the condition (256) it will be reset
```

For løkke, setter I til mindre en 256 og øker med en for hver gang opp til 255 og vil da nullstille og gå til 0 igjen.

```
{
```

Programmering Del 3

```
digitalWrite(slaveSelect, LOW); //Write our Slave select low to enable the SHift register to begin
listening for data
SPI.transfer(i); //Transfer the 8-bit value of data to shift register, remembering that the least
significant bit goes first
digitalWrite(slaveSelect, HIGH); //Once the transfer is complete, set the latch back to high to stop
the shift register listening for data
delay(delayTime); //Delay
}
}
```

`digitalWrite(slaveSelect, LOW);` Setter `slaveSelect` LAV for å angi at den skal hente data
`SPI.transfer(i)` overfører den 8 bit digitale verdien med valgte signifikante data først (i er en
8-bit integer. LSB eller HSB er satt av oss. Når data er overført settes «sperren» på igjen ved
at `slaveSelect` settes HØY og stopper søk etter data (Stopper klokken). Pausen `delayTime`
settes for hvor høy tellehastigheten skal være i denne binærtelleren.

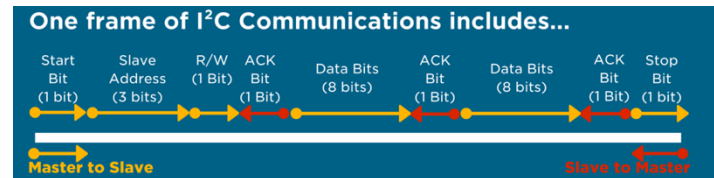
Kople opp og test programmet, endre forsinkelsen til eks. 250-400 ms, så du bedre kan se hva som skjer.
Endre også LSB til HSB så den teller motsatt.

I²-C Interface

For å kommunisere med sensoren benytter vi en meget populær bus som kalles I²C (Inter-Integrated Circuit) -uttales I-squared-C.

I²C har følgende egenskaper:

- Det er en seriell bus
- Master slave (Multi master Multi slave)
- Packet switched
- Singel ended



Inter Integrated Circuit, er noe tregere en SPI, men trenger bare 2 pinner for å sette opp seriell kommunikasjon, klokke og Data.

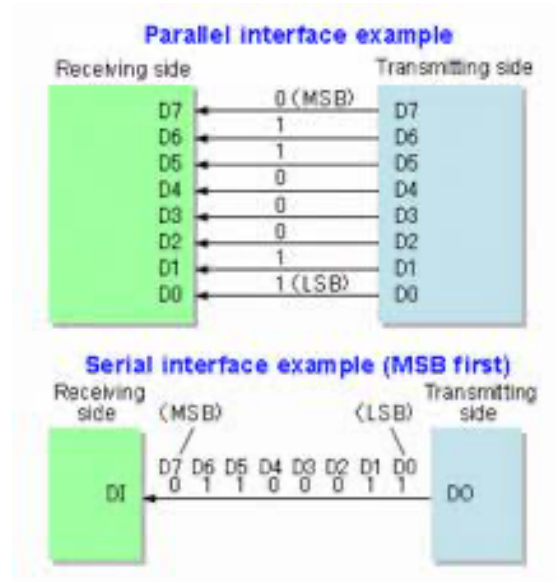
I²C er en halv Duplex protokoll, dvs. at den sender og mottar data på samme linjen, men ikke samtidig, slik den gjør i SPI protokoll.

Det finnes to typer busser seriell og parallell.

Parallelle busser bruker flere ledere til å overføre informasjon og brukes ofte når man kommuniserer over korte avstander. Fordi den har mange ledere så har den også høy kapasitet.

Serielle busser bruker en leder hvor man pakker informasjonen etter hverandre. Billig, robust (enkel å synkronisere). Blir brukt til mer og mer også over kortere distanser.

I²C protokollen kan styre og kontrollere hundrevis av enheter gjennom kun disse 2 pinnene.



Vi skal nå koble opp 2 Arduino UNO brett og samkjøre dem, et som master og et som slave. Kodene for de enkelte finner du i kodelinjen. Oppkopling finner du i oppkoplings linken.

[Link til oppkopling](#)

[Link til kode](#)

Programmering Del 3

Vi skal også benytte den serielle monitoren for å se på kommunikasjonen. La oss se over koden og forklare denne.

Master koden

```
#include <Wire.h>           // inkluderer biblioteket

void setup() {
  Wire.begin();             // delta på i2c bus (adressen er valgfri for master)
  Serial.begin(9600);       // start serial for output signal
}

void loop() {
  Wire.requestFrom(8, 6);   // ber om 6 bytes fra slave enhet #8

  while (Wire.available()) { // slave kan sende mindre en forespurt
    char c = Wire.read();   // motta byte som karakterer
    Serial.println(c);      // print karakterene på seriemonitoren vår
  }

  delay(500);              // legger inn en forsinkelse på 500ms.
}
```

Slave koden

```
#include <Wire.h>                                // inkluderer biblioteket

void setup() {
  Wire.begin(8);                                 // kople seg opp i2c bus med adressen #8
  Wire.onRequest(requestEvent);                 // denne funksjonen går «på» når “på” beskjed kommer fra master og
}

void loop() {
  delay(100);
}

// funksjon som utføres når data er forespurt fra master
// Denne hendelsen er registrert som en hendelse, se i setup.

void requestEvent() {

  Wire.write("hello ");                         // responder med en beskjed på 6 bytes
                                              // som forventet fra master
}
```

Kople opp og test i den serielle monitoren, skriver hallo en og en bokstav.

Kode for styring av RTC modul, DS 3231 fra Adafruit's

[Enheten koster rundt 150-160kr og kan bestilles her, husk å få med batteri i samme bestilling.](#)

[Last ned RTC biblioteket til din Arduino](#)

RTC er en “real time clock modul”.

```
// Date and time functions using a DS3231 RTC connected via I2C and Wire lib
#include <Wire.h>
#include "RTClib.h"      // Legger inn begge bibliotekene
RTC_DS3231 rtc;         // Den aktuelle chipen vi skal benytte

// Kalles for et to dimensjonalt array, 7, 12 definerer 7 enheter med 12 bokstaver
// som maks.
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"};
```

Programmering Del 3

```
void setup () {

  #ifndef ESP8266          // Angir en spesiell chip, men om den ikke er der initierer
                          // den seriell port.
    while (!Serial);     // for Leonardo/Micro/Zero
  #endif

  Serial.begin(9600);

  delay(3000);           // Vent 3 sek. for konsollet skal åpne

  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");    // Sjekker om rtc er koplet opp, gir
                                             // beskjed om den ikke er

    while (1);
  }

  // kalibreringsprosessen ved tap av spenningsforsyning
  if (rtc.lostPower()) {
    Serial.println("RTC lost power, lets set the time!");
    // following line sets the RTC to the date & time this sketch was compiled
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // This line sets the RTC with an explicit date & time, for example to set
    // January 21, 2014, at 3am you would call:
    // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
  }
}
```


Programming Del 3

```
void loop () {
    DateTime now = rtc.now();

    Serial.print(now.year(), DEC);
    Serial.print('/');
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.day(), DEC);
    Serial.print(" ");
    Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
    Serial.print(" ");
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.println();

    Serial.print(" since midnight 1/1/1970 = ");
    Serial.print(now.unixtime());
    Serial.print("s = ");
    Serial.print(now.unixtime() / 86400L);
    Serial.println("d");

    // calculate a date which is 7 days and 30 seconds into the future
    DateTime future (now + TimeSpan(7,12,30,6));

    Serial.print(" now + 7d + 30s: ");
```

Programmering Del 3

```
Serial.print(future.year(), DEC);  
Serial.print('/');  
Serial.print(future.month(), DEC);  
Serial.print('/');  
Serial.print(future.day(), DEC);  
Serial.print(' ');  
Serial.print(future.hour(), DEC);  
Serial.print(':');  
Serial.print(future.minute(), DEC);  
Serial.print(':');  
Serial.print(future.second(), DEC);  
Serial.println();  
  
Serial.println();  
delay(3000);  
}
```

Kople opp, test på seriell monitor. Ta av spenningsforsyningen og se om den resetter seg selv igjen.

[Link til DS3231 Bibliotek](#)

Bruk av interrupt og polling i Arduino program.

I denne delen skal vi se på hvordan du bruker (interrupt) avbrudd. Du har kanskje hørt om dem før, og de er en fin måte å øke programmets effektivitet når du arbeider med maskinvareinnganger. Polling er en funksjon som henter informasjon om eksempelvis bryterstatus. Polling sjekker regelmessig status på inngangen, noe som gjør at Polling ikke er så effektivt når programmet ditt får en hvis størrelse, da den avbryter programsekvensen for stadig å sjekke status.

Interrupt er en mer effektiv måte å gjøre dette på og er i praksis en hardware område på mikrokontrolleren som klarer å overvåke statusen på inngangen uten at den forstyrrer det andre som foregår. Den angir at det er en endring på inngangspinnen og varsler mikrokontrolleren om endringen (interrupt- bryter av) funksjon med små lavnivå instruksjoner.

Den fungerer på samme måte som å gå å vente på noen, du kan sjekke hele tiden eller få et signal når de er kommet.

Det finnes i prinsippet 3 forskjellige interrupt's:

- Change
- Falling
- Rising

Når du skal fastsette interrupt forteller du Arduinoen under hvilken fase du ønsker beskjed fra den om endringen. Falling, går fra høy til lav. Rising, går fra lav til høy eller Change som betyr at du ønsker beskjed uansett hva som skjeer. Når den får beskjed om aktuelle hendelse kjører den en spesifikk rutine knyttet til hendelsen. Denne kaller vi ISR, Interrupt service rutine. ISR koden bør være så kort som mulig på grunn av avbrytelsen, slik at den raskt kan gå tilbake til det den holdt på med.

Basis Interrupt, to do and dont do!

- Hold ISR- koden så kort som overhodet mulig
- ISR kan ikke returnere verdier eller parametere
- Delay og millis vil ikke fungere inne i ISR rutiner, da ISR er en delay i seg selv
- Delay micro sek vil kunne virke da den ikke er interrupt basert
- Enhver variabel inne i ISR skal angis med volatile modifier og det er kun enkelte pinner som har Interrupt mulighet.
- Arduino Uno har kun pinne 2 og 3

Programmering Del 3

- Det er ikke alle chiper som kan håndtere alle 3 interrupt typene.

La os se på et kodeeksempel

Vi bruke et standard bryteroppsett med 1 LED, likner på oppsettet vi brukte tidligere på if funksjonen.

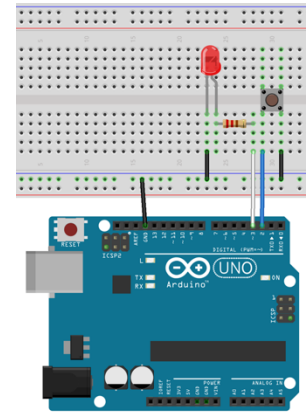
```
// pin definitions
int ledPin = 3;
int buttonPin = 2;

// global variables
int toggleState;
int lastButtonState = 1;
long unsigned int lastPress;
volatile int buttonFlag;           // Denne variable kan endres når som helst

int debounceTime = 20;

void setup() {
  // setup pin modes
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(2), ISR_button, CHANGE); // Overvåker endring på digital pinne 2(buttonPin).
  ISR_button er navnet på funksjone du ønsker.Change måler både opp og nedadgående endring skal registreres.
}

```



DigitalPinToInterrupt er den tryggeste syntaxen for Interrupt bruk, de finnes også andre.

```
void loop() {
  if((millis() - lastPress) > debounceTime && buttonFlag) //buttonFlag må være høy for å være sann og kan kun bli
  dette om ISR_button skifter til høy på pinne 2
  {
    lastPress = millis(); //update lastPress
    if(digitalRead(buttonPin) == 0 && lastButtonState == 1) //if button is pressed and was released last change
    {
      toggleState = !toggleState; //toggle the LED state
      digitalWrite(ledPin, toggleState);
      lastButtonState = 0; //record the lastButtonState,
    }
    else if(digitalRead(buttonPin) == 1 && lastButtonState == 0) //if button is not pressed, and was pressed last
    change
    {
      lastButtonState = 1; //record the lastButtonState
    }
    buttonFlag = 0;
  }
}

void ISR_button()
{
  buttonFlag = 0;
}
```

Programmering Del 3

Kople opp brettet og legg inn koden, last ned o test, vil fungere på lik linje med tidligere toggle switch system med if funksjonen.

Diverse linker

- [Arduino språk referanse](#)
- [Hackster.io/arduino](#)
- [Forum.arduino.cc](#)
- <https://www.keyestudio.com>
- [Arduino prosjekter- prosjekt HUB](#)
- <https://hackr.io/blog/how-to-learn-programming>
- <https://lastminuteengineers.com>

- <https://lastminuteengineers.com/nrf24101-arduino-wireless-communication/>
- <https://www.rapidtables.com/convert/number/binary-to-hex.html>
- <https://www.digikay.hu/en/maker/search-results?g=newest&t=1d5c49f5e90c43979b2d21f9036a92cd&y=13825c8674444e22884d8d26197819d1>
- <https://roboticsbackend.com/category/arduino/>
- <https://core-electronics.com.au>

Link til kode

Innhold Arduino sett

Nr	Stk	Tekst	Dato	Signatur ut	Retur enheter
1	1	Plastboks med innlegg			
2	1	Arduino UNO brett			
3	1	Utviklings brett 2560			
4	1	Breadboard hoved			
5	1	Mini Breadboard			
6	1	40 koplingspinner rett 2,54mm			
7	1	40 koplingspinner vinklet 2,54mm			
8	1	GP 10 utvidelsesbrett for Raspberry			
9	1	Akselrasjonssensor			
10	1	Ultrasonisk avstandssensor HC SR04			
11	1	Fjernkontroll, infrarød			
12	1	Ps2 Joystick			
13	1	Relekort 5V-10A			
14	1	Stepper motor 28 BYJ-48 5VDC			
15	1	Stepper motor driver modul			
16	1	RTC Modul (Real Time Clock)			
17	1	Justerbart potensiometer 1K			
18	1	Justerbart potensiometer 10K			
19	1	Aktiv buzzer 5V			
20	1	Passiv buzzer 5V			
21	1	LCD 1602			
22	1	8 segment x 1 tall			
23	1	8 segment x 4 tall			
24	1	8 x 8 matrix LED			
25	1	SG 90 Servo med tilbehør			

Programmering Del 3

26	1	Thermistor (temp avh motstand)			
27	1	IR Mottaker H x 1838			
28	1	Kulebryter			
29	1	LDR 5516 photoresistor			
30	1	SS 8050 NPN transistor			
31	1	LM 35 temperatur sensor			
32	1	Lyd sensor brett			
33	1	Infrarød sender			
34	1	Flamme sensor / photo transistor			
35	1	IC MAX 7219			
36	1	IC 74HC595- 8bit Shift Register			
37	1	RGB LED			
38	10	RØD LED			
39	10	GUL LED			
40	10	GRØNN LED			
41	5	Brytere med gult lokk			
42	1	USB-2 kabel			
43	10	Dupon kabel hun-han			
44	10	Dupon kabel hun-hun			
45	10	Motstand 220 ohm			
46	10	Motstand 1K ohm			
47	10	Motstand 10K ohm			
48	28	Korte kabler Rød, sort, blå, grønn, gul, hvit, Orange			
49	6	Lange 1xRød, 2xSort, 1XBlå, 2xHvit			
50	1	9V batteri adapterkabel			

Sjekkpunkter, hva har jeg lært?

- ____ Å kunne bruke variabler i en matrise.
- ____ Kunne bruke aritmetiske funksjoner, relasjonsoperatører, logiske og tildelingsoperatører.
- ____ Manipulere data ved hjelp av bit-vise operatører og logikk(boolsk)
- ____ Lagre informasjon mellom resetting ved hjelp av EEPROM
- ____ De forskjellige operatørene, tabeller
- ____ Skiftregistre, SIPO og PISO
- ____ SPI interface og I²O interface
- ____ Serielle og paralelle busser
- ____ Master og slave oppkopling
- ____ RTC modul, oppkopling og testing
- ____ Interrupt og Polling