Science Talent Rumteknologi

Vejledning til Arduino og sensorer, version 4

*science talenter

Indholdsfortegnelse

1. Ultrakort introduktion til Arduino

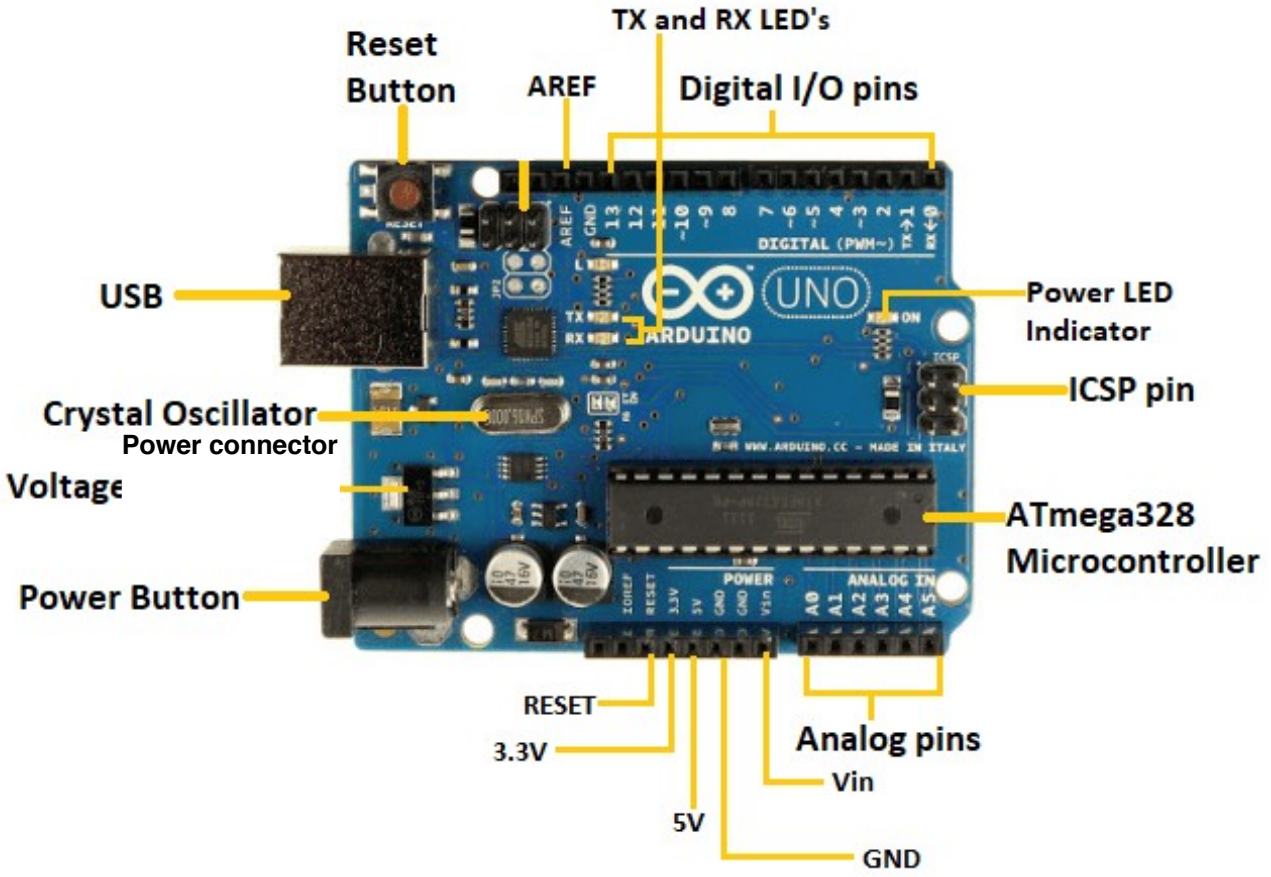
Har du allerede styr på, hvad en Arduino er, kan du gå videre til trin 2.

Om bord på satellitten eller rumfartøjet har vi brug for en computer, som kan

- modtage kommandoer fra Jorden,
- foretage målinger ved at aflæse sensorer,
- evt. sortere i data, og vurdere hvilke der skal sendes til Jorden, og hvilke der eventuelt skal kasseres, og selvfølgelig
- sende måledata og housekeepingdata til Jorden.

Som regel vælger man altid det simpleste system, der kan løse opgaven, og for mange cubesats og nanosatellitter vælger man typisk en simpel mikrocontroller, bestående af CPU, hukommelse, lager, og noget grundlæggende input-output-hardware.

Her bruger vi Arduino UNO eller Nano, som er bygget op omkring en AVR-mikrocontroller fra firmaet Atmel, som hedder ATmega328P. Den indeholder både en 8-bit 16 MHz CPU, 32 kB flash hukommelse, og en masse input/output-pins. Desværre indeholder den i sig selv ikke et lager, hvor programmet kan ligge, et USB-interface, strømforsyning eller en ”programmer”, der kan tage imod ny software og gemme det i lageret. Derfor har man valgt at kombinere alle disse ting i en ”Arduino”.


TX and RX LED's
Reset Button
AREF
Digital I/O pins
USB
Power LED Indicator
ICSP pin
Crystal Oscillator
Power connector
Voltage
ATmega328 Microcontroller
Power Button
RESET
3.3V
5V
GND
Analog pins
Vin
UNO
ARDUINO
DIGITAL (PWM~)
POWER
ANALOG IN

2. Download og installation af Arduino IDE

Har du allerede installeret Arduino IDE, kan du gå videre til trin 3.

En mikroprocessor arbejder med kommandoer, som gør forskellige ting, og som har hver sin talkode. Det sprog, processoren forstår, hedder maskinkode. Et program i maskinkode kunne f.eks. se sådan her ud:

```
E1 F0 6B 46 58 5C FD D1 B3 D5 86 4C 70 80 88 95
```

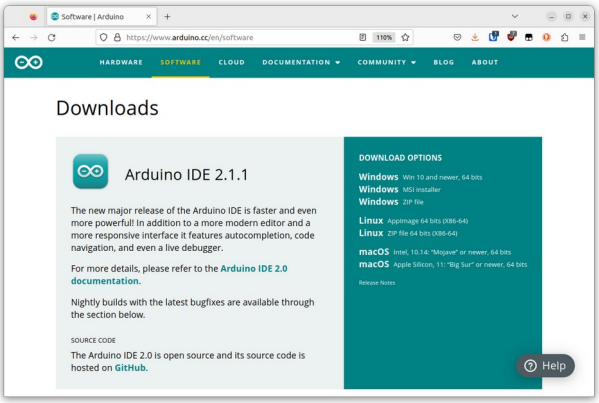
Det får vi mennesker ikke så meget ud af at se på. Derfor kalder vi maskinkode for et ”lavniveausprog”, og definerer så selv nogle ”højniveausprog”, som vi bedre kan forstå. Et program i højniveausprog kunne f.eks. se sådan her ud:

```
void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(500);
}
```

Her er det lidt nemmere at følge med i, hvad der sker uden et intimt kendskab til den konkrete processortype. Når vi har skrevet et program i et højniveausprog, har vi altså brug for en ”oversætter”, der kan oversætte programmet til maskinkode. Det kaldes en ”compiler”.

For at programmere en Arduino har vi altså brug for en teksteditor, hvor vi kan skrive programmet, en compiler, der kan oversætte programmet til maskinkode, og en USB-interface, så vi kan uploade det compilerede program til Arduinoen. ”Arduino IDE” er sådan et program – det indeholder både teksteditor, compiler, uploader og en masse ”libraries” og eksempler, så det altid er nemt at prøve noget nyt. IDE står for ”Integrated Development Environment.

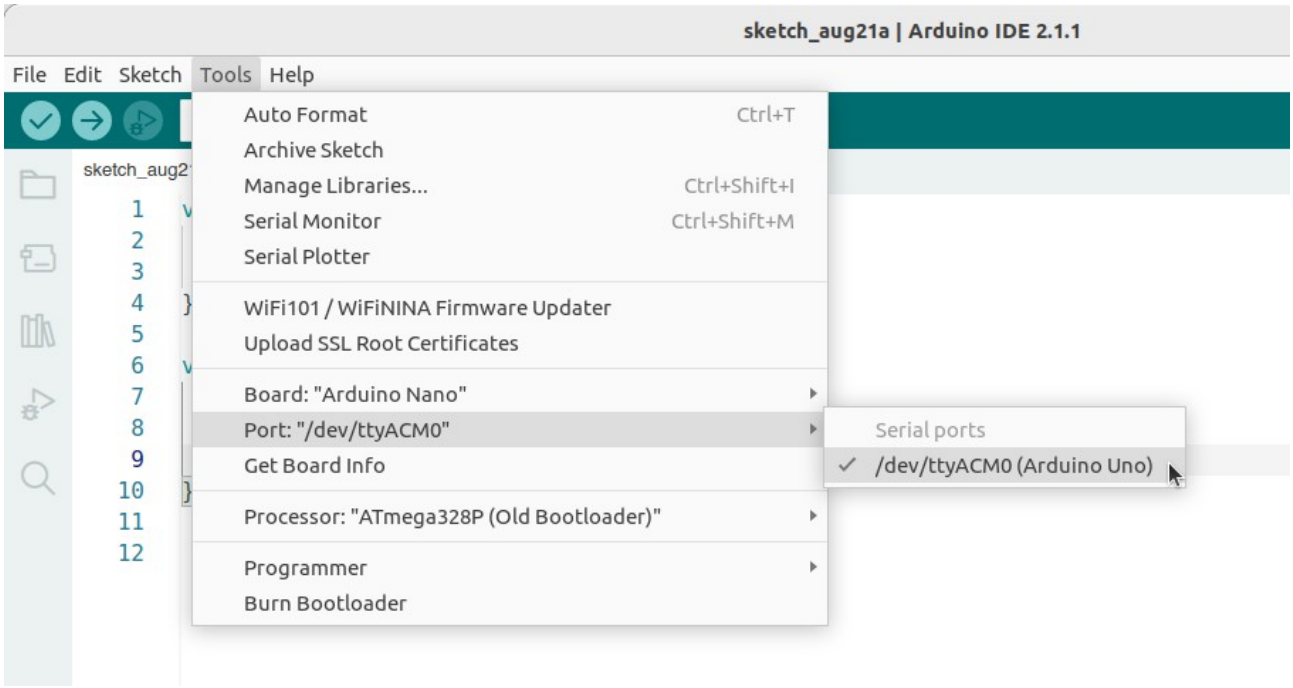
Gå ind på www.arduino.cc. Klik på ”Software” i topmenuen, og vælg den download, der passer til din computer: Installer programmet.



3. Upload dit første program

Har du allerede prøvet at uploade programmer til en Arduino, kan du gå videre til trin 4.

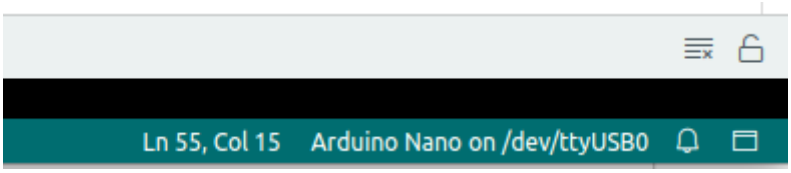
Slut Arduinoen til din computer med et USB-stik, og start Arduino IDE. Det er vigtigt, at Arduino IDE ved, hvilken port din Arduino er tilsluttet – det indstiller du her:



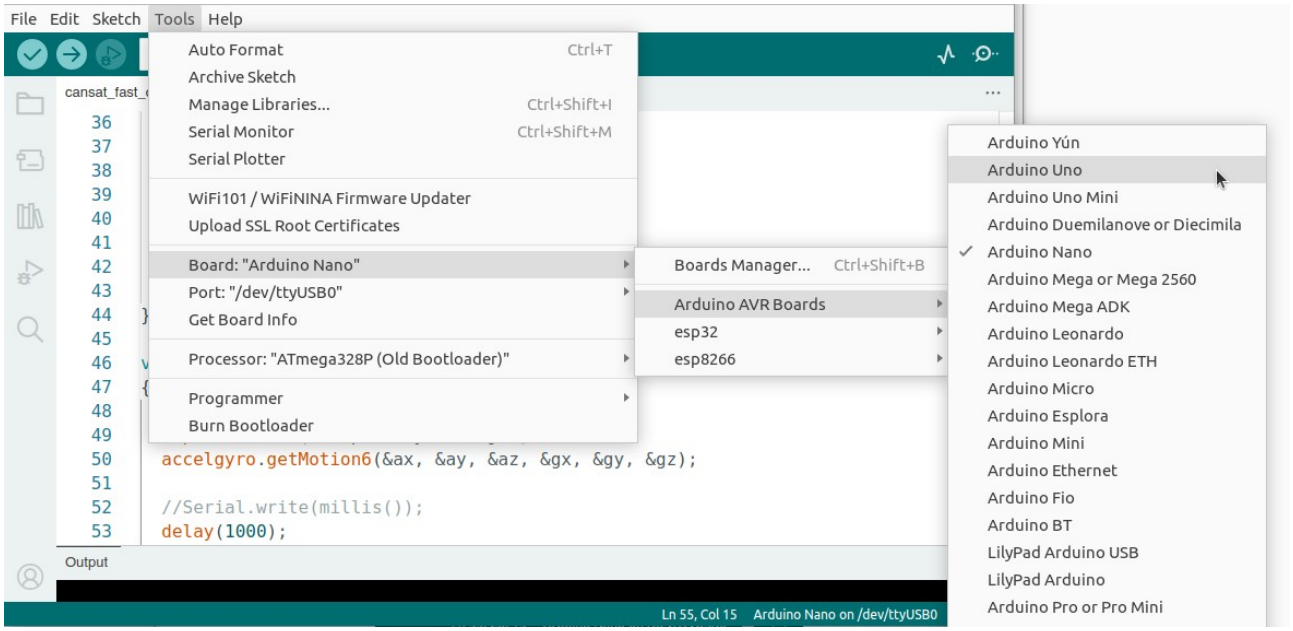
På Windows-computere hedder porten typisk `COM1` og opefter, men som regel står der "Arduino" ved den rigtige port. På Mac og Linux hedder porten som regel `/dev/ttyACM0` hvis det er en Arduino Uno, eller `/dev/ttyUSB0`, hvis det er en Arduino Nano.

> **OBS** Nogen gange (desværre ret ofte) mister Arduino IDE forbindelsen til Arduinoen. Når det sker, plejer den mest hensigtsmæssige procedure at være
> 1. Prøv at vælge en anden port i menuen,
> 2. Træk USB-stikket ud og sæt det i igen,
> 3. Genstart Arduino IDE
> 4. Alle de ovenstående

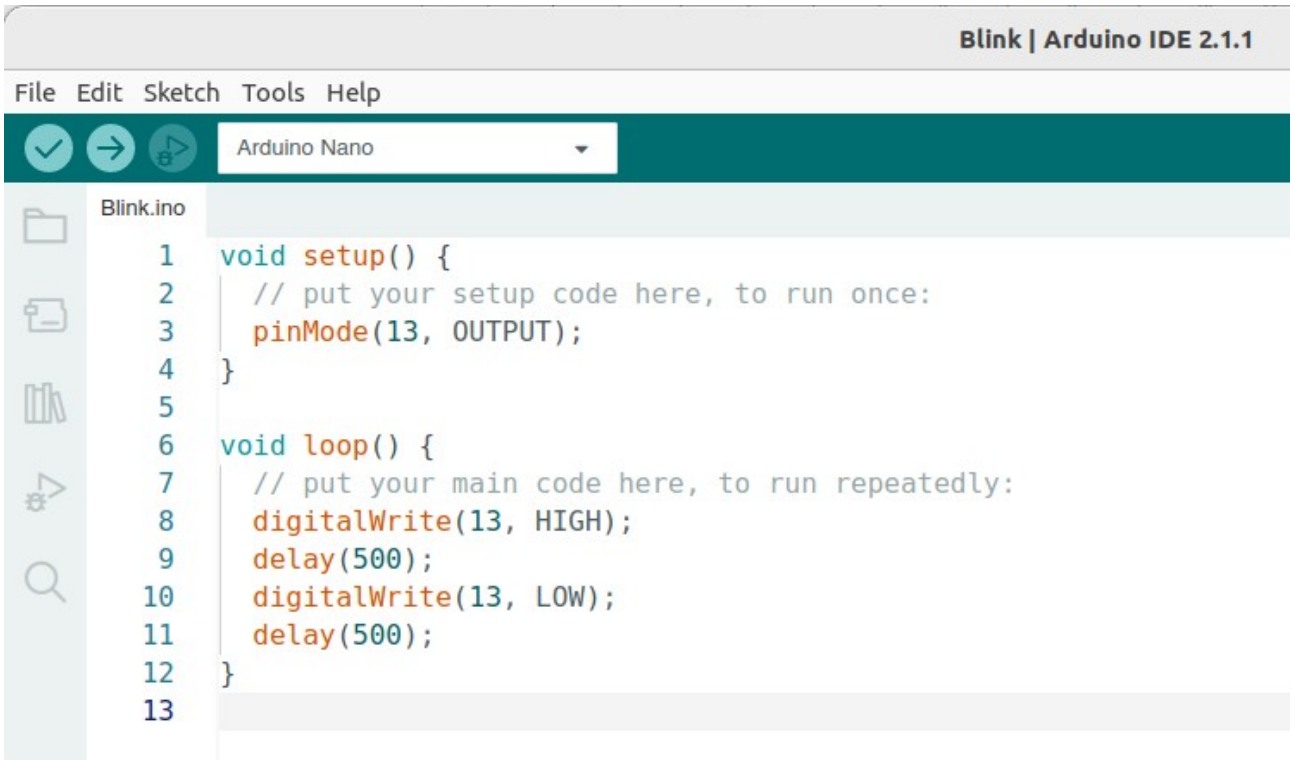
Du kan altid tjekke, om det korrekte board og den korrekte port er valgt ved at kikke i nederste, højre hjørne:



Hvis der ikke står, om det er en Arduino Uno eller Nano eller hvad du nu har, der står, kan du vælge det under Tools ➜ Board ➜ Arduino AVR Boards:



Brug teksteditoren til at skrive følgende program:



Kommentarer
Som du næsten kan regne ud, markerer to skråstreger efter hinanden ”kommentarer”, dvs. tekstuelle beskrivelser af, hvad denne del af programmet gør. Compileren ignorerer automatisk alle kommentarer. Det er god stil at kommentere sin kode, så andre, der skal bruge den, ikke behøver sidde og gætte på, hvad du har tænkt.

void, setup() og loop()
Det næste, du lægger mærke til er, at der er to ”blokke” i programmet; de begynder begge to med ”void”, den ene hedder ”setup”, og den anden hedder ”loop”. Disse to blokke skal altid være i et Arduino-program – hvis du fjerner en af dem, virker programmet ikke.

Når Arduinoen starter, køres det, der står i ”setup” fra toppen og nedefter. Derefter køres det ikke igen, før Arduinoen genstarter.

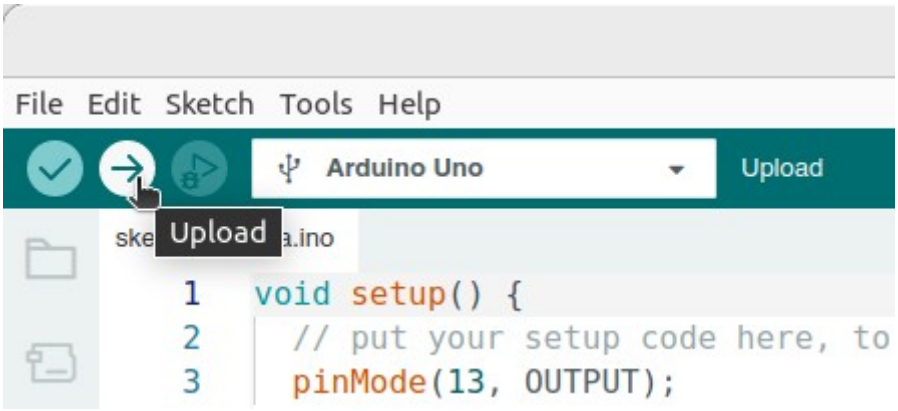
Når ”setup” er færdig, køres det, der står i ”loop”. Du kan næsten selv regne ud, at når ”loop” er færdig, begynder den forfra på det, der står i ”loop”. Og det bliver den ved med, så længe den har strøm, eller nogen trykker på resetknappen.

Husk semikolon!
Læg også mærke til, at næsten alle linjer (men ikke alle!) slutter med et semikolon. Det fortæller Arduinoen, at ”her var en kommando, og den skal du udføre nu”. De eneste linjer,

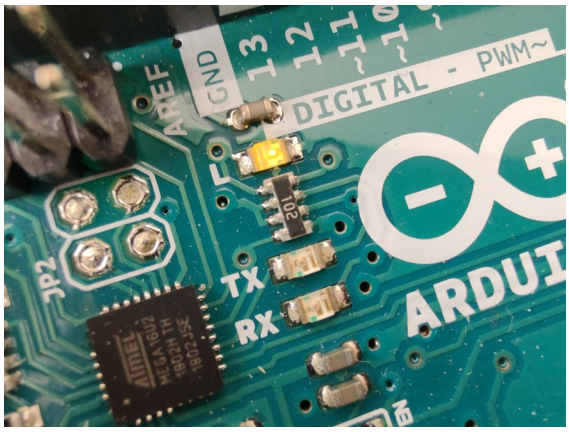
der ikke skal afsluttes med semikolon, er dem, der markerer, at ”her begynder en ny blok”, dvs. linjer, der slutter med en ”start-tuborg”, {.

> **OBS** Læg mærke til, at teksten imellem tuborgklammerne er rykket to mellemrum ind på linjen. Det gør det meget nemmere at overskue, hvad der foregår i et program, når der er mange af sådanne blokke inden i hinanden. Arduinoen er ligeglad med mellemrummene, men hvis du ikke er konsekvent med dine indrykninger, kan du godt gå hen og blive ked af det i det lange løb!

Upload
Du compiler og uploader programmet til Arduinoen ved at klikke på pilen øverst til venstre, eller ved at trykke ctrl-U:



Hvis programmet virker, vi du se den gule lysdiode, der hedder ”L” blinke:



Hvis programmet ikke virker, kan det skyldes forskellige ting:

- Hvis du har lavet en fejl i programmet, er compileringen ikke gennemført, og du vil se en fejl i bunden af Arduino IDE.

- Hvis der ikke er forbindelse til Arduinoen, vil der typisk også være en fejlmeddelelse et eller andet sted.

Hvad gør programmet?
Lad os hurtigt gennemgå de kommandoer, programmet indeholder.

I "setup" står der

```
pinMode(13, OUTPUT);
```

pinMode (stavet på den måde – med lille p og stort M) fortæller Arduino, om en af de 13 pinde, der hedder "DIGITAL (PWM~)", som både kan bruges til input eller output, skal være det ene eller det andet. Her siger vi til Arduino, at vi gerne vil bruge pin nr. 13 som en output-pin, dvs. vi vil selv kunne styre, om spændingsforskellen mellem pin 13 og jord/nul/stel skal være +5V eller 0V. Pin 13 er forbundet til lysdioden "L".

Når vi har sat pin 13 til output, kan vi give en ny kommando:

```
digitalWrite(13, HIGH);
```

digitalWrite (stavet på den måde – med lille d og stort W) fortæller Arduino, at vi gerne vil "skrive" en digital værdi til pin 13. En digital værdi kan være 0 eller 1, slukket eller tændt, +5V eller 0V, eller, som det hedder på Arduino-sprog, LOW eller HIGH. Her "tænder" vi for pin 13.
Hvis vi vil have lysdioden til at blinke, er det nærliggende at tænke, at vi skal slukke for lysdioden, lige så snart vi har tændt for den. Husk dog, at processoren i en Arduino Uno kører med 16 MHz, så i det tilfælde vil den tænde og slukke for lysdioden flere millioner gange i sekundet, og det kan vi ikke se. Derfor bliver vi nødt til at have den treje kommando:

```
delay(500);
```

delay (med lille d) angiver simpelthen et antal millisekunder, som Arduinoen skal vente, før den foretager sig noget som helst andet. Det giver fin mening her, hvor vi skal have en lysdiode til at blinke, men generelt er det en dårlig ide sådan at "spilde computerens tid".

4. Byg dit første kredsløb

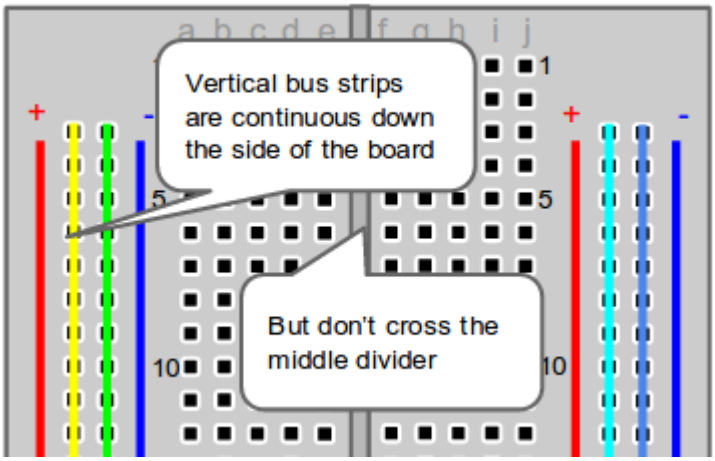
Har du allerede prøvet at bygge kredsløb på et protoboard, kan du gå videre til trin 5.

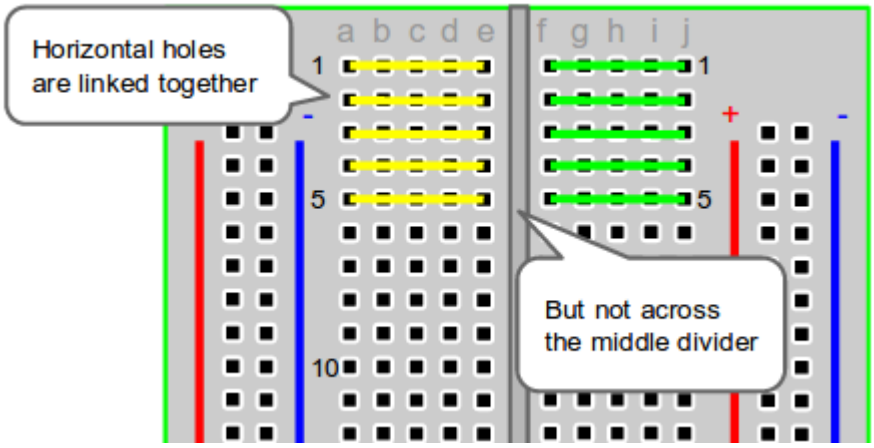
Du har nu fået en lysdiode til at blinke på Arduinoen – hurra! Arduinoen kommer dog først for alvor til sin ret, når den bruges til at interface med anden hardware, f.eks. sensorer. Derfor skal vi i gang med at forbinde den til noget eksternt. Helt grundlæggende kan man bruge Arduinoen til to ting:

1. at styre, kontrollere eller regulere ting
2. at måle, føle eller "sanse" omverden

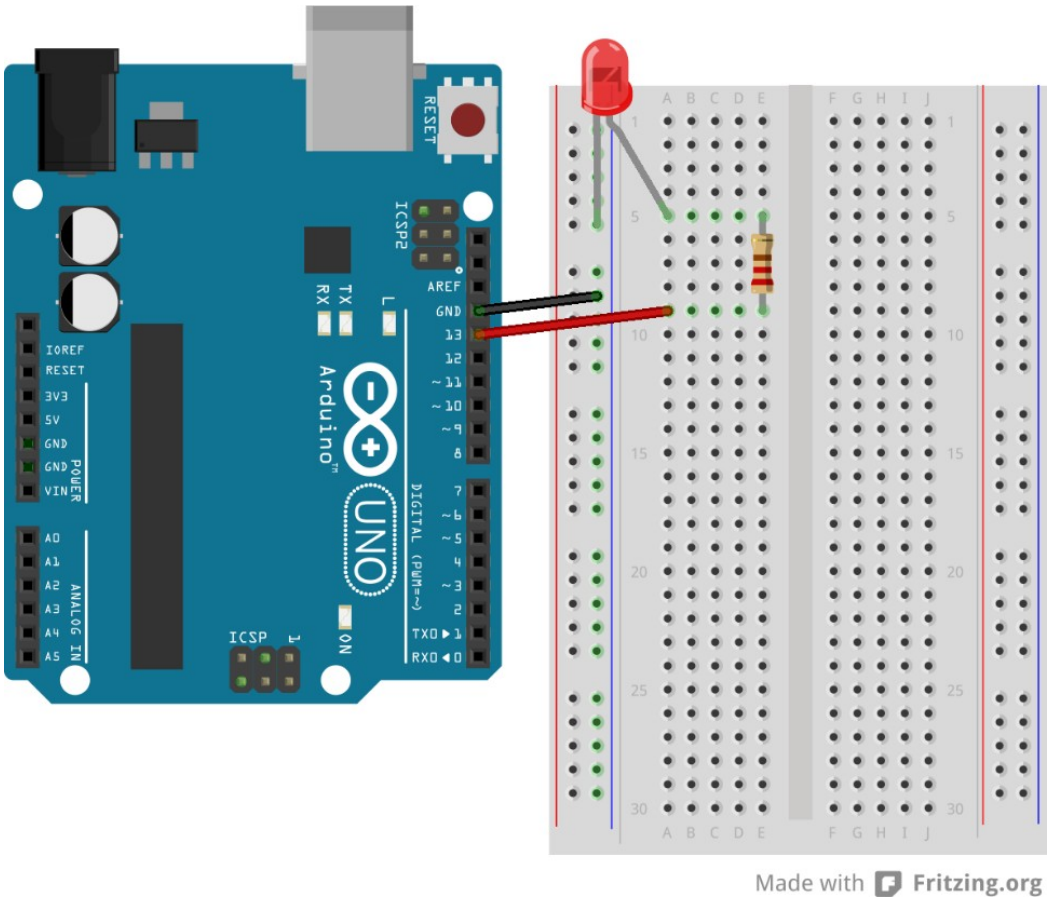
og rigtig smart bliver det, når Arduino kan måle noget, og derefter regulere noget, så en fysisk størrelse, f.eks. et belysningsniveau, en temperatur, en pH-værdi, en retning eller position i rummet kan fastholdes inden for nogle fastsatte rammer, ellers varieres i tid efter et ønsket mønster.

Protoboard
Vi starter med et uhyre simpelt kredsløb på et "protoboard", også kaldet "breadboard" eller "fumlebræt". Princippet i et protoboard er ret enkelt:





På den måde kan vi f.eks. opbygge et simpelt kredsløb med en Arduino, en lysdiode, en modstand og to ledninger:



Som det ses, bygger vi et kredsløb, hvor strømmen først ”kommer ud” af pin nr. 13 på Arduinoen, og dernæst løber igennem den røde ledning over til protoboardet. Dernæst løber strømmen igennem modstanden, som skal have værdien 220 Ω (ohm), farvekode rød-rød-brun. Så løber strømmen gennem protoboardet og hen til lysdioden, hvor det løber ind ad lysdiodens længste ben – en lysdiode kan kun virke i én retning. Strømmen løber ud ad lysdiodens korte ben, gennem protoboardet, og via den sorte ledning tilbage til

GND på Arduinoen, som står for ”ground”, og som også kaldes ”jord” eller ”stel”, og som kan betragtes som Arduinoens strømforsynings minuspol.

Hvis du ellers har det program kørende på Arduinoen, som vi lavede i afsnit 3. Upload dit første program, burde lysdioden blinke. Hvis ikke, så tjek, om den vender rigtigt. Hvis den stadig ikke virker, så prøv med en ny lysdiode.

Hvad skal modstanden til for?
Modstanden sikrer, at der ikke løber for stor strømstyrke igennem lysdioden. En lysdiode har utrolig lille intern modstand, og lægger vi bare en spændingsforskel på 5 V over en lysdiode, giver den et meget kraftigt og utrolig kortvarigt lysglimt fra sig, og så er den lige til at smide ud. Vi skal derfor sikre, at der ikke løber for stor strøm igennem den.

En anden – og måse vigtigere! – ting er, at der også er en klar grænse for, hvor meget strøm vi kan trække ud af en Arduino, før processoren brænder over. En Arduino koster lidt mere end en lysdiode, så her går vi lidt mere i detaljer:[1]

```
Atmel ATmega328P (UNO and Duemilanove) Current Specifications:
Absolute Maximum Ratings - the point where damage will start to happen
DC Current per I/O Pin ..... 40.0 mA
DC Current VCC and GND Pins..... 200.0 mA
1 VCC pin: Means these Arduinos can Source a total of 200 mA
2 GND pins: Means these Arduinos can Sink a total of 400 mA
```

Ohms lov
Vi må altså maksimalt trække 40 mA (helst kun 30 mA) ud af en Arduino-pin, som har en spænding på 5 V. Hvor stor skal modstanden så være? Her skal vi bruge Ohms lov, som siger, at

$$U = R \cdot I$$

hvor U er spændingsforskellen, R er modstanden og I er strømstyrken:

$$U = RI$$
$$R = \frac{U}{I} = \frac{5\ \mathrm{V}}{30\ \mathrm{mA}} = \frac{5\ \mathrm{V}}{0{,}030\ \mathrm{A}} = 167\ \frac{\mathrm{V}}{\mathrm{A}} = 167\ \Omega$$

167 Ω-modstande findes ikke – derfor vælger vi den nærmeste, højere værdi, som er 220 Ω.

1 https://playground.arduino.cc/Main/ArduinoPinCurrentLimitations/

5. Seriel kommunikation og variable

Har du allerede prøvet at sende data over en serielforbindelse, kan du gå videre til trin 6.

Hvis vi bruger vores Arduino til at måle noget, så skal vi gerne have de målinger ud af Arduinoen, og over på vores computer. Lige nu er vores computer forbundet med Arduinoen med et USB-kabel, men det kunne i praksis lige så godt være en trådløs radioforbindelse.

Indtast og upload dette program:

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

void loop() {
  Serial.println("Hej med dig");
  delay(500);
}
```

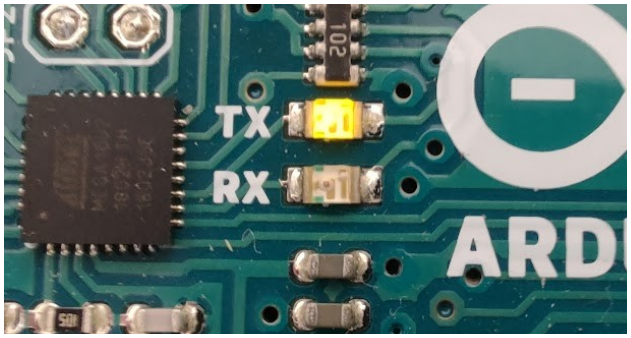
I setuppet angiver vi, at vi gerne vil starte en transmission af serielle data fra Arduinoen og ud på USB-kablet, og at vi gerne vil have den til at sende med en hastighed på 9600 bits pr. sekund.[2]

I loopet sender vi en tekststreng, ”Hej med dig”, ud på serielforbindelsen. Vi kunne også have brugt kommandoen `Serial.print("Hej med dig");`, men så havde vi ikke fået et linjeskift med til sidst, og som vi skal se, er det vældig praktisk med sådan et.

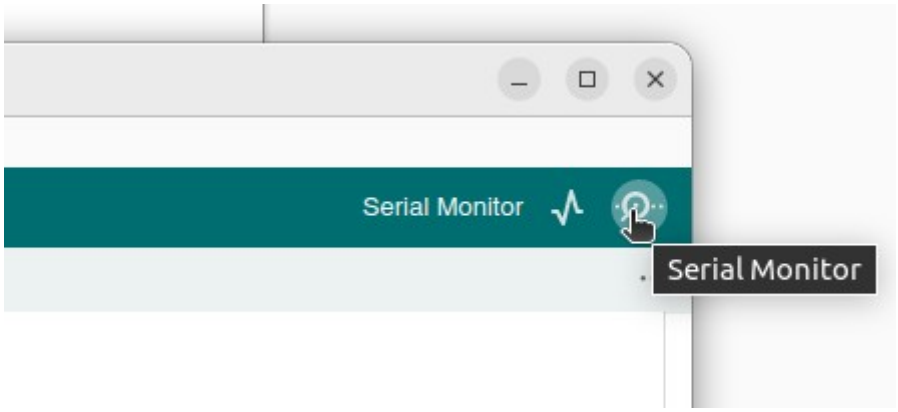
Til sidst et lille delay, så vi ikke oversvømmer serielforbindelsen fuldstændigt.

Når programmet er uploadet, er det svært at se, at der sker noget – bortset fra, at lysdioden TX begynder at lyse gult:

2 Hvorfor lige 9600 bit/s? Det har historiske årsager: I tidernes morgen valgte man at bruge standardhastigheden 75 bit/s til fjernskrivere (eng. ”teletype”, også kaldet TELEX). Så fordoblede man til 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, og til sidst 57600, som var det maksimale, man i midten af 90’erne kunne klemme igennem et parsnoet kobberkabel. Disse hastigheder er blevet hængende i forbindelse med seriel kommunikation.



For at se de data, Arduinoen sender tilbage til computeren åbnes en serielmonitor: Tryk på denne knap øverst til højre:



-eller CTRL-SHIFT-M. Nu skulle der gerne dukke et lille vindue op, hvor teksten ”Hej med dig” bliver printet på en ny linje to gange i sekundet. Samtidig begynder lampen TX at blinke i takt med transmissionerne.

Tillykke! Nu kan din Arduino kommunikere med omverden!

OBS Nogle gange ”dør” serialmonitoren, og der bliver ikke vist nye værdier, selvom nye værdier bliver sendt fra Arduinoen. Løsningen er næsten altid at lukke serialmonitoren og åbne den igen.

Man behøver sådan set ikke have Arduino IDE installeret for at aflæse, hvad der modtages på serielporten – man kan downloade ”stand-alone”-serielmonitorer, eller man kan f.eks. selv aflæse den i et Python-program, som så kan gemme data eller manipulere dem til hvad man har lyst til, f.eks. gemme dem i en database, tegne fine grafer, etc.

Nu er det ikke så spændende at se den samme tekststreng hele tiden, så prøv f.eks. med

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

int i=0;

void loop() {
  Serial.println(i);
  i++;
  delay(500);
}
```

Her definerer vi en variabel, som vi kalder `i`. Den har typen `int`, som betyder et heltal i intervallet [−32768; 32767].

Teknisk set er der tale om et 16-bit tal, dvs. et tal, der kan repræsenteres af 16 bits, dvs. 2^{16} = 65536 forskellige værdier. Man vælger så at fordele lægge disse 65536 værdier, så man kan angive både positive og negative værdier – dette kaldes en ”signed integer” (heltal med fortegn). Hvis man hellere vil have heltal mellem 0 og 65535, skal man bruge typen `unsigned int`.

Kommandoen `int i=0;` ”instantierer” variablen, og sætter samtidig dens værdi til nul.

Kommandoen `i++;` forøger værdien af variablen med 1. Hvis variablen allerede har værdien 32767, vil den efter denne kommando få værdien −32768.

Scope
Hvorfor står kommandoen `int i=0;` mellem `setup()` og `loop()`?

Hvis den stod inde i loopet, ville vi sætte værdien af variablen til 0 hver gang, og den ville aldrig kunne forøge sin værdi.

Hvis den stod inde i setuppet, ville vi få en fejl, når programmet nåede ned i loop, fordi variablen har et ”scope”, der svarer til det ”niveau”, hvor den er defineret. Hvis vi definerer en variabel i setuppet, kan vi kun bruge den i setuppet. Hvis vi definerer den i loopet, kan vi kun bruge den i loopet. Hvis vi definerer den udenfor både setup og loop, kan vi bruge den globalt, dvs. overalt.

Forskellige variabeltyper

`int`	16-bit heltal mellem −32768 og 32767
`unsigned int`	16-bit heltal mellem 0 og 65535
`long`	32-bit heltal mellem minus og plus 2,1 milliarder
`string`	en streng af tekst, f.eks. ”Hej med dig.”
`bool`	en binær værdi, som kun kan have to forskellige værdier, `true` eller `false` (sand eller falsk).

OBS Når du skal til at sende flere data af sted på samme tid, er det en god ide at "komma-separere" dem. Hvis du f.eks. foretager samtidige målinger af lufttryk og -temperatur, er det nærliggende at sende dem af sted sådan her:

```
Lufttryk: 1008.75 hPa
Temperatur: 21.7 °C

Lufttryk: 1005.23 hPa
Temperatur: 21.2 °C

Lufttryk: 1002.16 hPa
Temperatur: 20.7 °C
```

Men typisk vil man gerne have sine data ind i et regneark, så man kan tegne nogle fine grafer – og et output som det ovenstående er super svært at hive ind i Excel, særligt hvis man måler i længere tid. Derfor er det en rigtig god ide at have en enkelt linje i sit setup, der skriver f.eks.:

```
messagenumber;milliseconds;pressure;temperature
```

og så efterfølgende

```
324;2055;1008.75;21.7
325;3058;1005.23;21.2
326;4061;1002.16;20.7
```

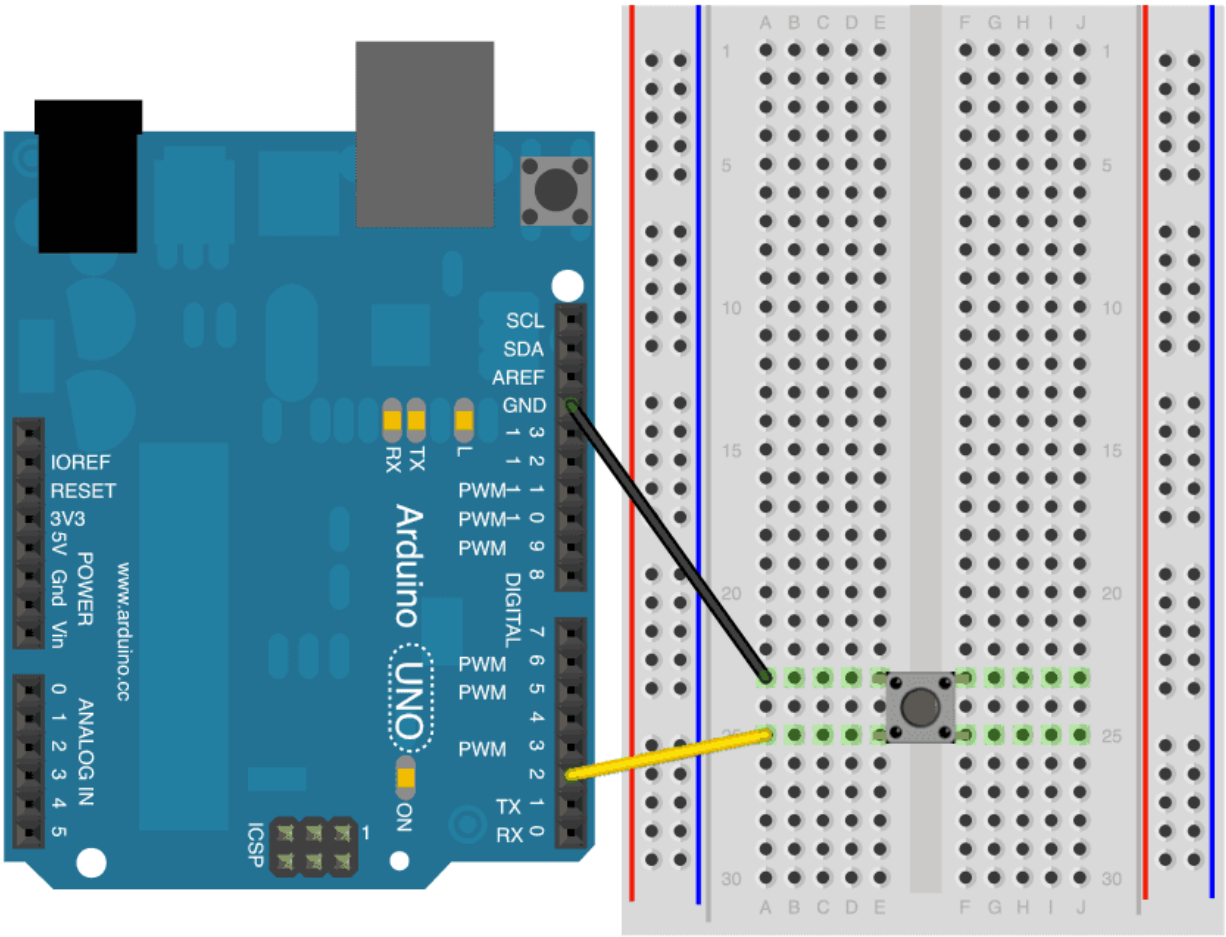
Her printes alle målinger på en linje, så det er nemt at importere ind i et regneark. Her har vi også valgt at sende et "message number", som bare er et fortløbende tal – det gør det nemt at se, hvis man har mistet data – og antallet af millisekunder siden sidste boot, som ligger i variablen millis()

Læg også mærke til, at selvom vi kalder det "komma-separerede værdier" (eng. Comma Separated Values – CSV) – typisk med filendelsen *.csv – så bruger vi her semikolon i stedet. Det skyldes, at nogen regneark laver decimalpunktummerne om til kommaer, og hvis vi så også har separeret vores værdier med komma, er vi på den!

6. Aflæsning af digitale sensorer og `if`

Har du allerede prøvet at aflæse en digital sensor, kan du gå videre til trin 7.

Den mest grundlæggende form for sensor er en, der kan føle, om der er spænding på en indgang eller ej. F.eks. kan vi opbygge sådan et kredsløb her, med en trykknap og to ledninger:



Vi forbinder pin 2 til knappens ene side, og knappens anden side forbinder vi til GND. Hvis vi definerer pin 2 som en inputpin, kan vi detektere, om vi trykker på knappen, fordi pin 2 så bliver forbundet til GND:

```
void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT);
}

void loop() {
  int sensorVal = digitalRead(2);
  Serial.println(sensorVal);
}
```

Her åbner vi en serielforbindelse, definerer pin 2 som INPUT-pin, så nu kan vi ikke bruge kommandoen `digitalWrite` til at skrive til den, men kommandoen `digitalRead` til at læse fra den. Det gør vi i loopet, hvor vi definerer en `int`, som hedder `sensorVal`, og tildeler den den værdi, der kommer ud af pin 2, hvorefter vi skriver den til serielforbindelsen.

Når du opbygger kredsløbet som vist og uploader programmet, skulle du gerne se en række værdier i serialmonitoren, som enten er 0 eller 1, eller står og skifter imellem 0 og 1. Når du så trykker på knappen, skulle du gerne kun se nuller. Når du slipper knappen igen, begynder den at skifte mellem 0 og 1 igen, eller også ser du bare stadig nuller.

Her melder to spørgsmål sig hurtigt – lad os tage det nemmeste først:

1. Hvorfor skriver den 0, når vi trykker på knappen, og ikke 1?

Når man bruger kommandoen `digitalRead`, beder man dybest set arduinoen måle spændingsforskellen mellem den pågældende pind (her pin 2) og GND. Hvis spændingen måles til mellem 0 og 0,8 V, tæller det som et nul. Hvis spændingen måles til mellem 2,2 og 5 V, tæller det som et ettal. Hvad så, hvis spændingen ligger et sted der i mellem? Ja, så bliver arduinoen i tvivl, og det er mere eller mindre tilfældigt, om resultatet bliver et ettal eller et nul.

Når vi trykker på knappen, forbinder vi pin 2 direkte til GND, så et voltmeter mellem de to vil måle 0 V. Derfor udlæses 0, når vi trykker på knappen.

Når vi ikke trykker på knappen, er pin 2 ikke forbundet til noget – hverken til 0 V eller til 5 V. Derfor sender arduinoen enten skiftevis nuller og ettaller – hvis spændingen ligger og svinger et sted derimellem – eller stadig lutter nuller. Og nu har vi næsten også svaret på det andet spørgsmål:

2. Hvorfor skriver den *ikke* 1, når vi *ikke* trykker på knappen?

Ja, det er så fordi, pin 2 ikke er forbundet til 5 V. Det kan man gøre selv, via en modstand, f.eks. 10 kΩ – dette kaldes en ”pull-up-modstand”, fordi den ”trækker” input-pinden ”op”

mod 5 V, når man ikke trykker på knappen. ELLER man kan bruge arduinoens indbyggede pull-up-modstand – dette gøres, når man erklærer pinden som input-pind:
`pinMode(2, INPUT_PULLUP);`

Prøv selv – nu kan vi skelne imellem, når vi trykker på knappen, og når vi ikke gør!

Næste skridt – `if`

Her kunne det være rart, hvis arduinoen kunne gøre noget særligt, hvis vi trykker på knappen, og noget andet, hvis vi lader være. Hvis du stadig har lysdioden siddende på pin 13, så prøv f.eks.

```
void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT_PULLUP);
  pinMode(13, OUTPUT);
}

void loop() {
  int sensorVal = digitalRead(2);
  Serial.println(sensorVal);

  if( sensorVal == 1 ) {
    digitalWrite(13, LOW);
  } else {
    digitalWrite(13, HIGH);
  }
}
```

`if( sensorVal == 1 )` spørger "er sensorværdien lig med 1?" Hvis ja, så udføres kommandoen – eller kommandoerne – i den efterfølgende blok omsluttet af tuborgklammer. Hvis nej, springer programmet videre. Der behøver ikke være en `else`, men hvis der er, udføres det, der står i den efterfølgende blok kun, hvis udsagnet i `if` var falsk.

Der er selvsagt ingen grænser for, hvor meget man kan have stående mellem tuborgklammerne.

7. Analoge sensorer og spændingsdelere

Har du allerede prøvet at aflæse en analog sensor, kan du gå videre til trin 8.

I sidste afsnit så vi, hvordan vi kan bede arduinoen fortælle os i hvilket af to spændingsintervaller, en pin er. Men hvad nu, hvis vi gerne vil have lidt finere opløsning? Hvad hvis vi f.eks. ikke bare vil vide, om en knap er nedtrykket eller ej, men hvor langt den er nedtrykket, hvor hårdt der bliver trykket på den, eller hvor langt den er drejet?

Heldigvis har Arduino indbygget seks analoge indgange, `A0`–`A5`, som er tilsluttet en 10-bit ADC. ADC står for Analog-to-Digital Converter, og kan, som navnet antyder, omsætte en analog spænding til en af 2^{10} = 1024 forskellige værdier, dvs. et tal mellem 0 og 1023.

Prøv at indtaste følgende, simple program:

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

void loop() {
  int sensorVal = analogRead(A2);
  Serial.println(sensorVal);
}
```

Programmet aflæser simpelthen den analog indgang A2, og skriver resultatet til serielporten. Hvis du ikke har forbundet A2 til noget, skulle du gerne se værdierne stå og svinge imellem 0 og 1023. Og det gør den selvfølgelig, fordi vi endnu ikke har forbundet den til noget. Hvis du forbinder A2 med en ledning direkte til GND, skulle den gerne vise 0. Hvis du forbinder den direkte til 5V, skulle den gerne vise 1023. Og forbinder du den til 3.3V, skulle du gerne få en værdi omkring 3,3V / 5,0V * 1023 = 675.

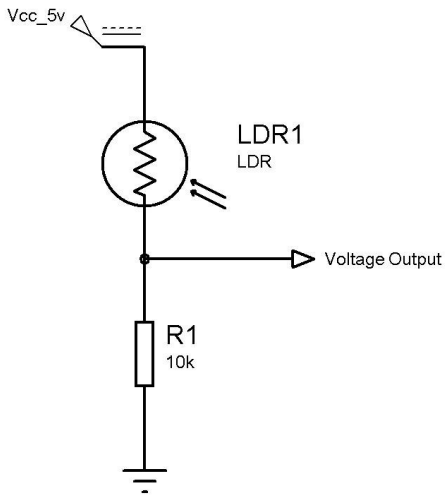
Der er mange forskellige typer sensorer, nogle af de mest almindelige er f.eks.:

- et potentiometer, der kan måle, hvor stor en vinkel, man drejer den,
- en termistor, der kan måle en temperatur, eller
- en LDR (Light Dependent Resistor), også kaldet en fotoresistor, der måler en lysstyrke.

Alle disse tre sensorer måler en fysisk størrelse ved at have en variabel modstand afhængig af den fysiske størrelse. Når vi ved, at sensoren virker som en variabel

modstand, kan vi godt se, at det ikke duer at slutte f.eks. en termistor direkte til 5V og A2. Uanset værdien af modstanden vil A2 være forbundet til 5V, og den udlæste værdi vil være 1023. Vi bliver nødt til at bygge en spændingsdeler, før vi kan bruge vores analoge sensor.

Spændingsdeleren
Kig på diagrammet herunder. Vi har serieforbundet to modstande: Den variable modstand, her en lysmodstand, som måler lysniveauet, og en fast modstand med kendt værdi, her 10 kΩ. Lysmodstanden er forbundet til 5V i den ene ende, lysmodstandens andet ben er forbundet til 10 kΩ-modstanden, og 10 kΩ-modstandens andet ben er forbundet til GND. Det punkt, hvor de to modstande mødes, kaldes midtpunktet, og det er dette punkt, vi forbinder til vores analoge indgang, f.eks. A2.



Husk på, at når vi udlæser en analog indgang med kommandoen `analogRead`, så beder vi faktisk arduinoen måle spændingsforskellen mellem midtpunktet og GND.

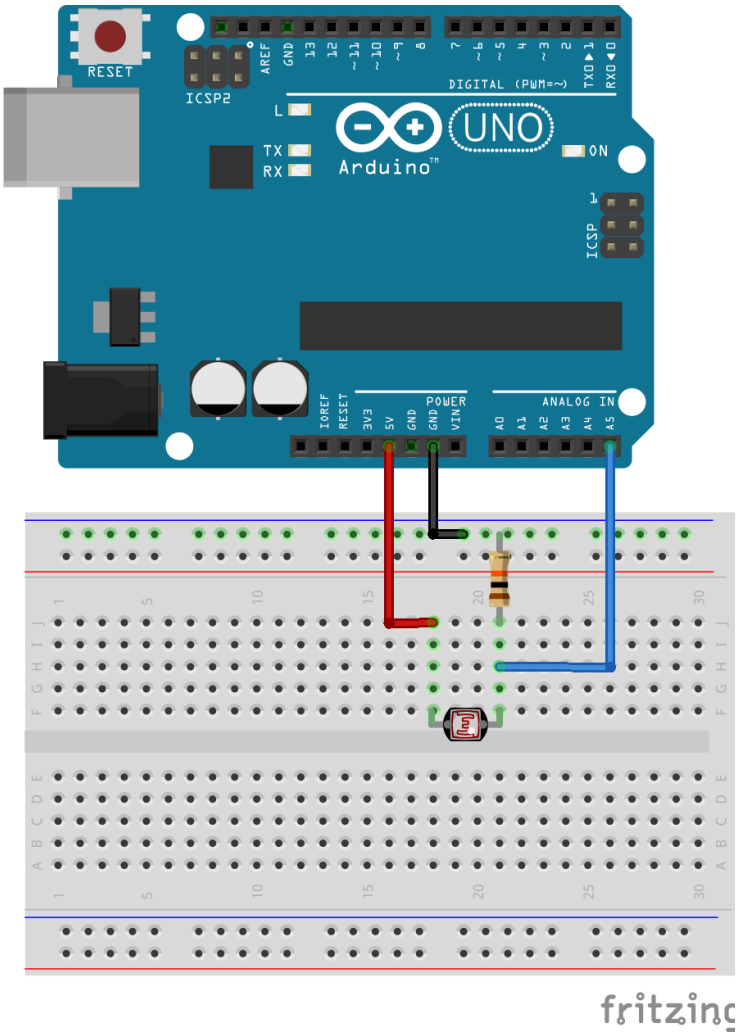
Hvis vi forestiller os, at den variable modstand, her LDR1, bliver meget lille – endda nul – så vil midtpunktet i princippet være direkte forbundet med 5V, og vi vil måle en værdi på 1023.

Hvis vi forestiller os, at den variable modstand bliver meget stor – endda nærmest uendelig – så vil midtpunktet i princippet være forbundet til GND, og vi vil måle en værdi på nul.

Hvis de to modstande er lige store, f.eks. 10 kΩ hver, så vil spændingsfaldet mellem 5V og GND være ligeligt fordelt over de to modstande, og vi vil måle en spænding på 2,5V eller ca. 512.

Det siger næsten sig selv, at hvis ens variable modstand kan variere mellem 10 og 20 Ω, så er en 10 kΩ-modstand alt for stor! Hvis den varierer mellem 500 og 800 kΩ, så er 10 kΩ klart for lidt. Så man skal vælge den faste modstand, så den passer til det område, ens sensor genererer.

Enough! Let us build!
Opbyg dette kredsløb på protoboardet:



(På tegningen er lysmodstanden sluttet til A5, men der er frit valg, så længe det er den samme port, du udlæser i programmet.)

Nu skulle du gerne se tallene i serialmonitoren ændre sig i takt med lysindfaldet, når du dækker for den.

Prøv i Arduino IDE at åbne Tools → Serial Plotter.

Prøv med en termistor, eller et potentiometer.

Udfordring
Byg et system, der tænder for en lysdiode, når det bliver for mørkt.

8. I²C

Har du allerede prøvet at interface til I²C-enheder, kan du gå videre til trin 9.

Somme tider vælger man lave lave en sensor – eller en hel pakke med sensorer – som en helt separat computer, man kan interface til. For at små computere, der sidder meget tæt på hinanden, kan tale med hinanden, har man udviklet protokollen I²C, på engelsk "eye-squared-see", på dansk som regel bare "i-to-se". Det er en "matematicering" af forkortelsen IIC, som står for "IC-to-IC Communication", hvor IC står for "Integrated Circuit" eller integreret kredsløb; det, vi på normalt dansk kalder en "mikrochip".

Mange sensorer og andre moduler bruger I²C, f.eks. de udbredte GY-87, GY-88, GY-91 eller HW-290. Her har man valgt at samle

- en Bosch BMP085 eller BMP180 lufttrykmåler (måler også temperatur),
- en TDK InvenSense MPU-6050 6-DOF (Degrees-Of-Freedom – "frihedsgrader") "Motion Tracker", bestående af et tre-akset accelerometer, der måler acceleration op til ±16 G i tre akser, samt et tre-akset gyroskop, der måler rotationshastigheder op til ±2000 °/s omkring samme tre akser,
- en Honeywell HMC5883L, HMC5983L eller QMC5883L, som indeholder et tre-akset magnetometer, der er i stand til at måle en magnetisk fluxtæthed på op til ±800 µT (mikrotesla – til sammenligning ligger jordfeltet på ca. 50 µT).

Her er tale om meget små enheder – på billedet er den metalskinnende klods trykmåleren, MPU-6050 er den største af de sorte "lakridser", og magnetometeret er den næststørste. Og det vel at mærke på et print, der måler i omegnen af 2x3 cm.

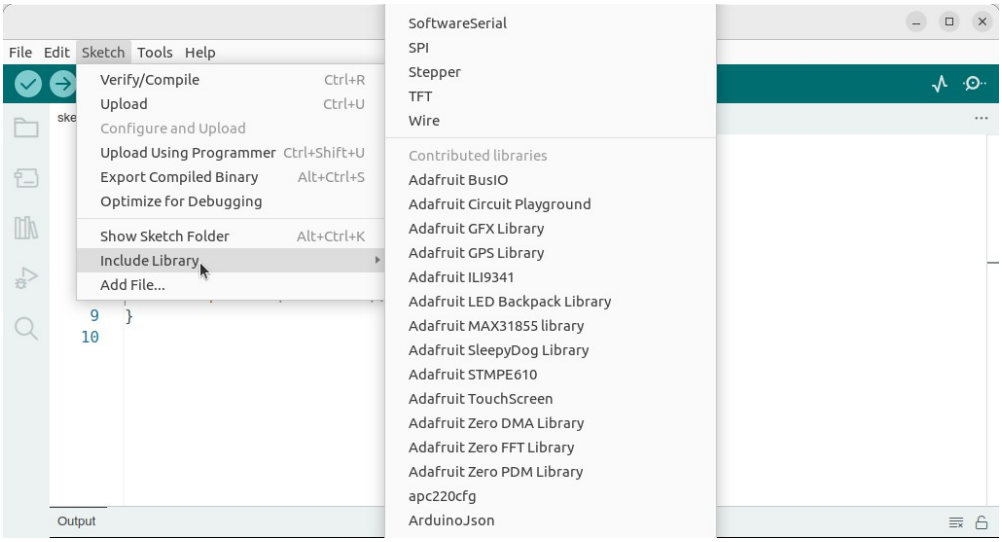
Der er tale om sensorer, som ofte sidder i almindelige mobiltelefoner, og derfor er blevet, og stadig bliver, produceret i enorme antal, og derfor er utrolig billige. Ombord på printet er også elektronik, der kan lave 5V om til 3.3V, og stå for I²C-kommunikationen mellem modulet og den omgivende verden.

I²C kræver to forbindelser: SCL og SDA – det står for Serial Clock og Serial Data. SDA sender dataene af sted fra I²C-enheden til arduinoen, og SCL er et tidssignal fra arduinoen, der siger til I²C-enheden, hvornår den må sende data.

Libraries

Hvordan man interfacer til I²C-enheder er typisk ret individuelt, og derfor er der næsten altid nogen, der har udviklet "libraries", som tilføjer nogle funktioner, der gør det nemmere at bruge enheden.

Hvis du går op under Sketch → Include Library får du en liste over libraries, du kan inkludere i din sketch, så du kan tilgå dine I²C-enheder.



Hvis du skal bruge et library, som ikke er på listen, skal du først tilføje det. I så fald skal du klikke på Sketch → Add File... og vælge et library i form af en zip-fil. I delemappen ligger arduino-libraries til BMP085, MPU6050, HMC5883 og QMC5883.

Til flere af disse skal du også have installeret librariet I2Cdev.zip, som også ligger i delemappen.

I mange tilfælde skal du også bruge librariet Wire, men det er allerede installeret i Arduino IDE fra starten af.

Stik GY-87'eren i et protoboard, og forbind dem som følger:

Arduino		**GY-87**
GND		GND
5V	forbindes til	VCC_IN
SCL		SCL
SDA		SDA

BMP180

Gå op i File → Examples → bmp085 → bmp085test. Upload sketchen, og åbn serialmonitoren. Du skulle gerne se en serie målinger af temperatur og lufttryk, og samtidig en omregning af lufttryk til højde. Vær opmærksom på, at denne omregning bliver foretaget af arduinoen baseret på et referencetryk, som er angivet med kommandoen `bmp085Init(reference_pressure)`

Ændr koden, så det eneste, der sendes over serielforbindelsen, er trykket i Pa – en måling pr. linje. Gå op under Tools → Serial Plotter og se, hvordan lufttrykket ændrer sig "live", når I flytter GY-87'eren op eller ned.

Ændr koden, så I kun får outputtet den beregnede højde. Brug det senest målte lufttryk i `bmp085Init`, så I får vi den relative højde i forhold til, hvor måleren er nu.

MPU-6050

Gå op i File → Examples → MPU6050 → accgyrotest. Upload sketchen og åbn serialmonitoren. Du skulle gerne se en serie målinger af accelerationer i x-, y- og z-aksernes retninger (ax, ay og az) samt målinger af rotationer omkring x-, y- og z-akserne (gx, gy og gz).

Ændr koden, så kun enten acceleration eller rotation outputtes på serielforbindelsen, og åbn Serial Plotter. Nu burde I kunne se værdierne ændre sig "live".

Hvis du i setuppet, efter `handleCalib()`, giver kommandoen

```
accelgyro.setFullScaleAccelRange(range);
```

hvor `range` er et tal mellem 0 og 3, vil det indstille accelerometeret til det tilsvarende måleområde:

range	måleområde
0	±2 G
1	±4 G
2	±8 G
3	±16 G

På samme måde kan du ændre følsomheden for gyroskopet med kommandoen

```
accelgyro.setFullScaleGyroRange(range);
```

hvor

range	måleområde
0	±250 °/s
1	±500 °/s
2	±1000 °/s
3	±2000 °/s

HMC5883L (på GY-87)
Gå op i File ➔ Examples ➔ HMC5883L ➔ test. Upload, og I skulle gerne kunne se data i serialmonitoren. Prøv at dreje sensoren og se, hvordan udlæsningerne ændrer sig.

QMC5883L (på HW-290)
Gå op i File ➔ Examples ➔ Mecha_QMC5883L ➔ raw. Upload, og I skulle gerne kunne se data i serialmonitoren. Prøv at dreje sensoren og se, hvordan udlæsningerne ændrer sig.

9. Avanceret styring og regulering

Nu er det efterhånden nærliggende at kombinere nogle af alle de sensorteknologier, vi har haft i spil, med noget styring eller regulering af en eller flere fysiske størrelser, baseret på sensorinputs.

Et klassisk eksempel på sensorbaseret regulering er skumringsrelæet, som mange husejere har i deres indkørsel eller carport: En lysmåler, der tænder for noget belysning, når det bliver mørkt. Det er dybest set rørende enkelt: En lysmåler, der tænder for en lampe, når belysningsniveauet kommer under et vist niveau.

Sådan et kredsløb kan I sagtens bygge med en Arduino nu.

Her kan man allerede ane konturerne af alle de problemer, der kan melde sig:

Uønsket feedback
Hvis lampen tændes, når det bliver mørkt, så er det jo ikke mørkt længere, for nu er lampen tændt. Hvis sensoren sidder for tæt på lampen, så vil den jo straks reagere ved at slukke lampen igen. Så bliver det mørkt, og så tænder lampen igen, og så videre. Det er jo ikke så smart.

Grænsetilfælde og hysterese
Et andet problem kan være, at lysniveauet ikke falder jævnt. Hvis solen er på vej ned, og der så går en sky for solen, så tænder lampen. Hvis skyen så går væk igen, så slukker den igen. Der kan således være en periode på måske en halv time hver morgen og aften, hvor lampen står og slukker og tænder hele tiden. Det er uhensigtsmæssigt, og kan løses ved at indføre ”hysterese” – at lampen tænder, når lysniveauet kommer under f.eks. 20 (arbitrær enhed), men ikke tænder igen, før det kommer over 30.

Overshoot og proportional regulering
Det er svært at regulere systemer med inerti. Opvarmning og afkøling er klassiske eksempler. F.eks. skal vi varme en ovn op, så lufttemperaturen inde i den bliver 200°C. Vi måler løbende temperaturen, og slukker for vores 2 kW-varmelegeme, når temperaturen når 200 grader, og tænder for det igen, når den kommer under. Men denne form for ”tænd-sluk-regulering” har sine problemer:

Vores varmelegeme når at blive meget varmere end 200 grader, når det har været tændt et stykke tid, så temperaturen i ovnen når op over 230, før den begynder at falde igen. I samme øjeblik temperaturen rammer 199 tænder vi igen, men fordi varmelegemet har en vis inerti, når temperaturen at falde til 180, før den begynder at stige igen.

Fænomenet kaldes ”overshoot”, og kan løses på flere måder, men den klassiske er at variere strømmen gennem vores varmelegeme, så den er proportional med afvigelsen fra den ønskede temperatur. Det er klart, at hvis temperaturen er større end vores måltemperatur, skal strømmen gennem varmelegemet være nul.

Men hvis det ikke er tilfældet, kan man beregne forskellen imellem vores måltemperatur og den ønskede:

$$\Delta T = T_{ønsket} - T_{målt}$$

Nu skal strømmen gennem varmelegemet være proportional med denne forskel:

$$I_{varmelegeme} = k\,\Delta T$$

Så vil vi varme kraftigt op, når vi er langt fra måltemperaturen, men jo tættere vi kommer på den, desto mindre vil vi varme.

Vi kender det også fra, når man kører bil – så slipper man ikke bare speederen, men letter trykket på pedalen lidt, efterhånden som man nærmer sig den ønskede fart.

En mere avanceret reguleringsteknik kaldes PID, for Proportional-Integral-Differential, hvor man også tager højde for integralet af målværdien (f.eks. hvis vi har kørt lidt for hurtigt i et stykke tid skal vi også køre lidt for langtsomt i et stykke tid, hvis vi skal ankomme til det rigtige tidspunkt) – og differentialkvotienten af målværdien (f.eks. hvis temperaturen begynder at stige meget hurtigt, skal vi måske skrue lidt ned, selvom vi stadig er langt fra målværdien, for ikke at forårsage overshoot.)

PID-regulering er noget, man behandler i kontrolteori, som er en hel teknisk videnskab i sig selv.

10. Forslag til avancerede projekter

Platform, der altid står vandret
Brug en MPU-6050 til at måle tyngdefeltet i x-, y- og z-retningen og to servo- eller steppermotorer til at bygge en platform, der altid vender vandret, uanset hvordan underlaget vender.

Udmåling af thrustkurve
Skriv software til en Arduino, som måler motorkraft som funktion af tiden for en raketmotor vha. en strain-gauge loadcell.

Lyssensor til raket
Mål rotationshastighed af en raket vha. en lyssensor (OBS – virker nok bedst i solskin!)

Automatisk vandingssystem til planter
Mål løbende jordbundsfugtigheden i en potteplante, og pump mere vand i, hvis den bliver for tør. (OBS – kræver enten god tid eller mange, små planter!)

Find selv på flere projekter!