Laboration 2 - Göra mätningar med en sensor

Avancerad elektronik och programmering

I den här labben fortsätter vi att bygga på de kunskaper vi fått i första labben gällande programmering och elektronik. När man deltar i en CanSat tävling ska man utföra några uppdrag och framförallt ska man åtminstone ha ett primärt uppdrag (eng. primary mission) där man mäter temperatur och lufttryck. I den här labben får du lära dig att använda en sensor som gör just detta.

När man programmerar är det väldigt noga att koden man skriver blir rätt. Datorer är generellt inte smarta nog att förstå när något är ett misstag. Detta är särskilt viktigt i väldigt dyra projekt som t.ex. uppskjutningen av ESAs Cluster-satelliter[1] vilken misslyckades på grund av en variabel var fel definierad.

Material

- Arduino MKR ZERO
- Kopplingsplatta
- USB-sladd för att koppla Arduino till datorn
- Dator
- Diod
- Resistor
- Adafruit BMP280
- Sladdar för koppling

[1] https://www.inf.ed.ac.uk/teaching/courses/seoc/2008_2009/resources/ariane5.pdf

E-post: cansat@au.se
Telefon: 070-000 90 56
Senast uppdaterad: 3/10-22

Funktioner

En Arduino programmeras med ett språk som är baserat på ett programmeringsspråk som kallas C. C är ett språk som till stor del bygger på funktioner. Vi har redan stött på ett par inbyggda funktioner, både `digitalWrite` och `delay` är funktioner. En funktion anropas (används) genom att man skriver dess namn följt av en parentes. I parentesen skriver man funktionens argument, d.v.s. vilka variabler eller värden som skickas till funktionen när den startar. Om funktionen inte har några argument lämnas parentesen tom, om den har fler än ett argument separeras de med kommatecken.

> **Inforuta, kom ihåg:** `delay(ms)`
> `delay()` är en funktion som pausar programmer på den raden i ms millisekunder.

Vi kan titta närmare på `digitalWrite` som ett exempel. Det är en funktion som skickar ström till en vald pin. Funktionen skrivs som

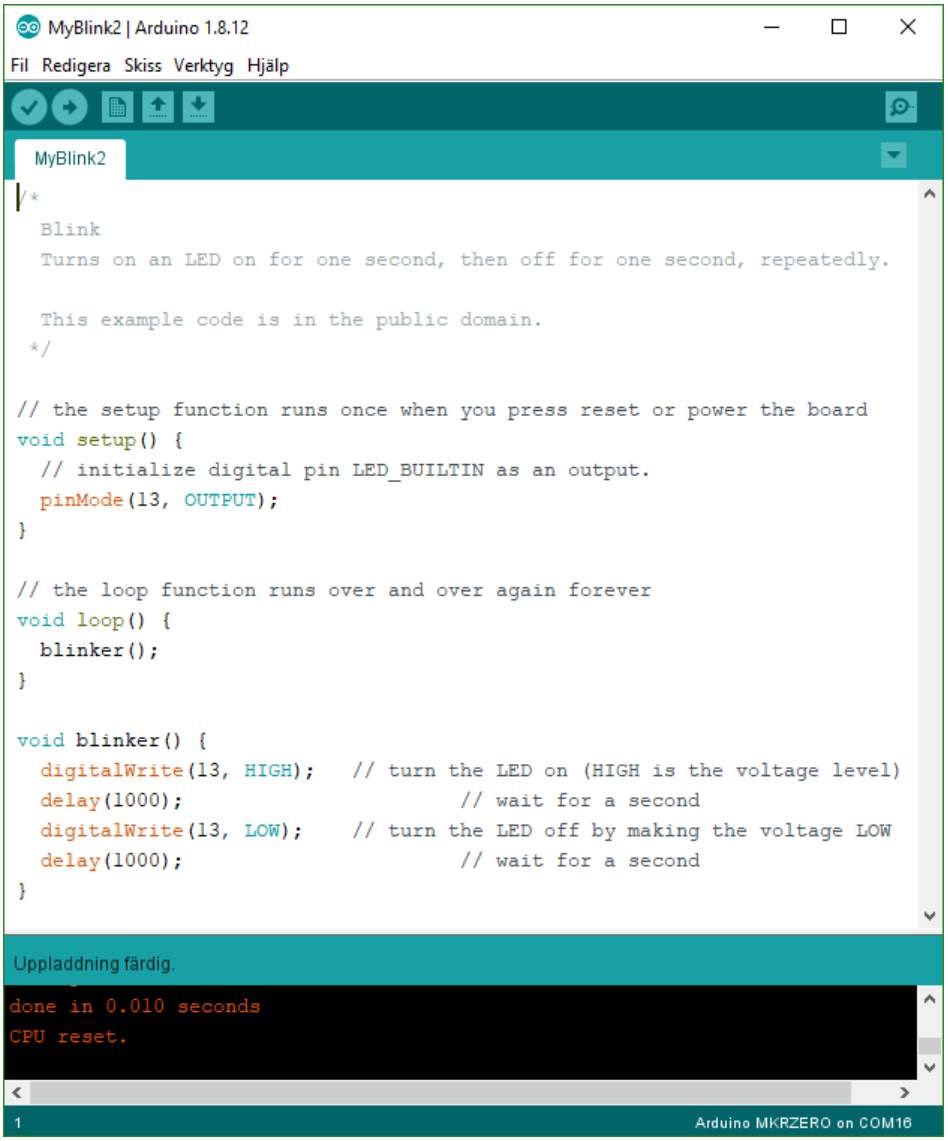
- `digitalWrite (pin, value);`

där *`pin`* är namnet på den pin som vi använder och *`value`* är antingen HIGH eller LOW. LOW betyder 0V och HIGH betyder 5V eller 3,3V beroende på vilken Arduino man har. För `pin` kommer generellt anges en siffra som motsvarar en pin man använder. Denna typ av information om alla inbyggda funktioner finns i Arduinos referens[2].

Man kan även skriva egna funktioner. Vi man se exempel på det i koden vi redan har. `void setup() {...}` och `void loop() {...}` är funktioner. När man definierar en funktion i C måste man först skriva vilken typ av data funktionen kommer leverera, båda våra exempel kommer inte ge någon data alls så ordet `void` (tom) används. Det betydet att funktionen kommer inte göra något mer än den kod som står innanför klamerna. Det andra ordet är funktionens namn. Både `setup` och `loop` är speciella namn som är reserverade för just dessa funktioner. `setup` körs först när programmet startar och bara en gång. Koden i `loop` körs om och om igen. Parentesen efter namnet används för data som ska föras in i funktionen, om ingen data ska in i funktionen lämnas parentesen tom. Sist kommer ett par klammerparenteser med all kod inom sig. Detta är koden som definierar vad funktionen gör.

---

[2] http://www.arduino.cc/reference/en



Vi ska nu skriva en egen funktion för att demonstrera detta. Vi utgår från exemplet MyBlink som vi använde i förra labben. Det absolut enklaste vi kan göra är att skriva en funktion som gör det som redan görs i `loop()` och anropa den i `loop()`. Funktionen ska skrivas utanför alla andra funktioner. Vi skriver

- `void blinker() {}`

Detta är funktionens definition klipp ut koden i `loop` och klistra in den mellan klammerparenteserna i `blinker` Resultatet ska se ut som ovan.

Om koden laddas upp nu borde den fungera som innan, men den nya strukturen låter oss göra fler saker. Men först måste vi diskutera lokala, globala och statiska variabler.

Variabler

Variabeln `waitTime` som vi introducerade tidigare är en så kallad global variabel. Det är den eftersom dess definition står utanför alla funktioner. Globala variabler kan användas i alla funktioner. Om man definierar en variabel inuti en funktion är den en lokal variabel för den funktionen och kan inte kommas åt utifrån funktionen. För att demonstrera detta ska vi ändra våran kod så att perioden mellan blinkningarna av lampan ökar med tiden.

> **Inforuta, kom ihåg:** Variabler
> När man skapar en variabel måste man alltid ange vilken datatyp den ska ha. Hittills har vi använt `int` (för heltal).

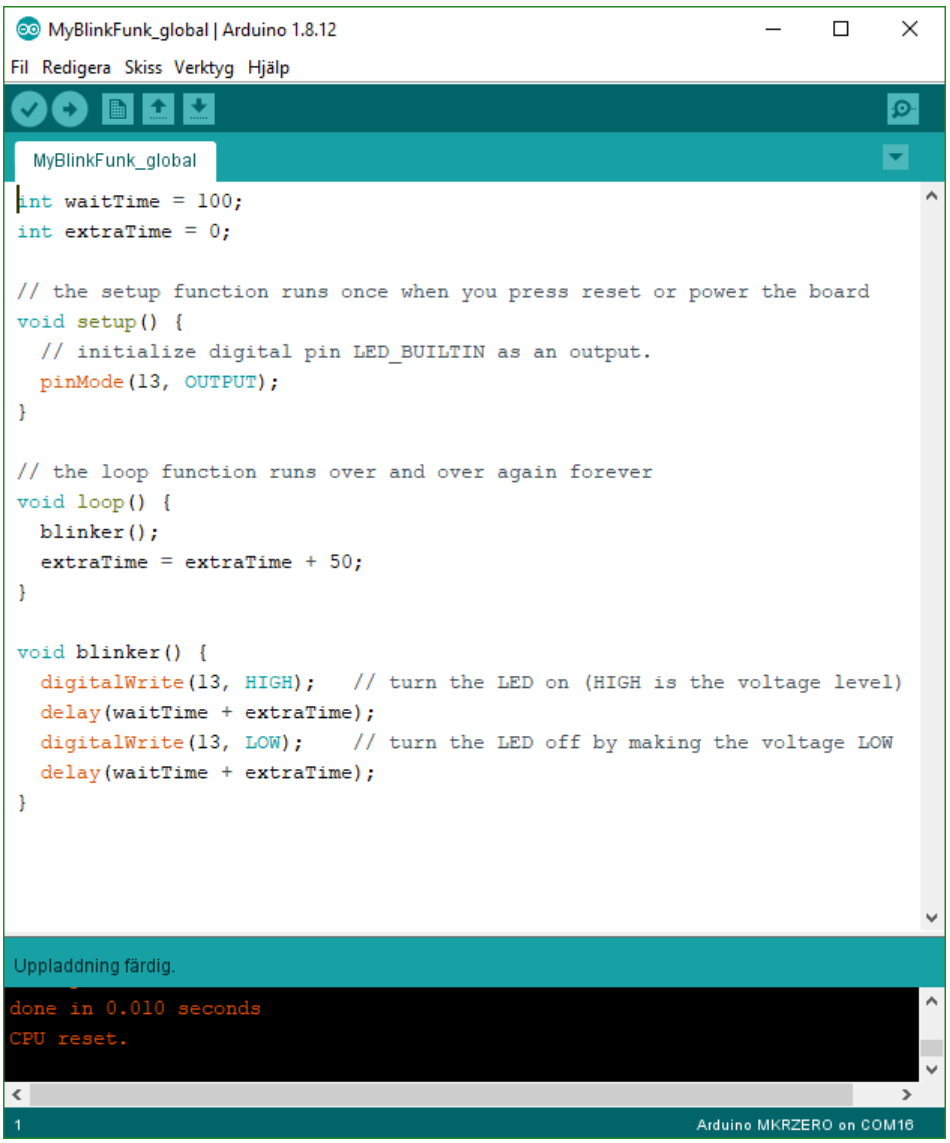
Globala variabler

Om vi vill använda globala variabler kan vi göra såhär. Skapar en ny variabel som heter `extraTime` med värdet `0`

- `int extraTime = 0;`

Placera den direkt efter vi definierar `waitTime`. Ändra argumenten i våra delay-funktioner till `extraTime + waitTime`. I `loop()`, efter den anropar `blinker()`, öka värdet på `waitTime` med `50`.

- `extraTime = extraTime + 50;`



Notera att alla variabeldeklarationer sker utanför funktioner och att de används fritt i olika funktioner. Nu ska vi göra samma sak fast med lokala variabler.

Lokala variabler

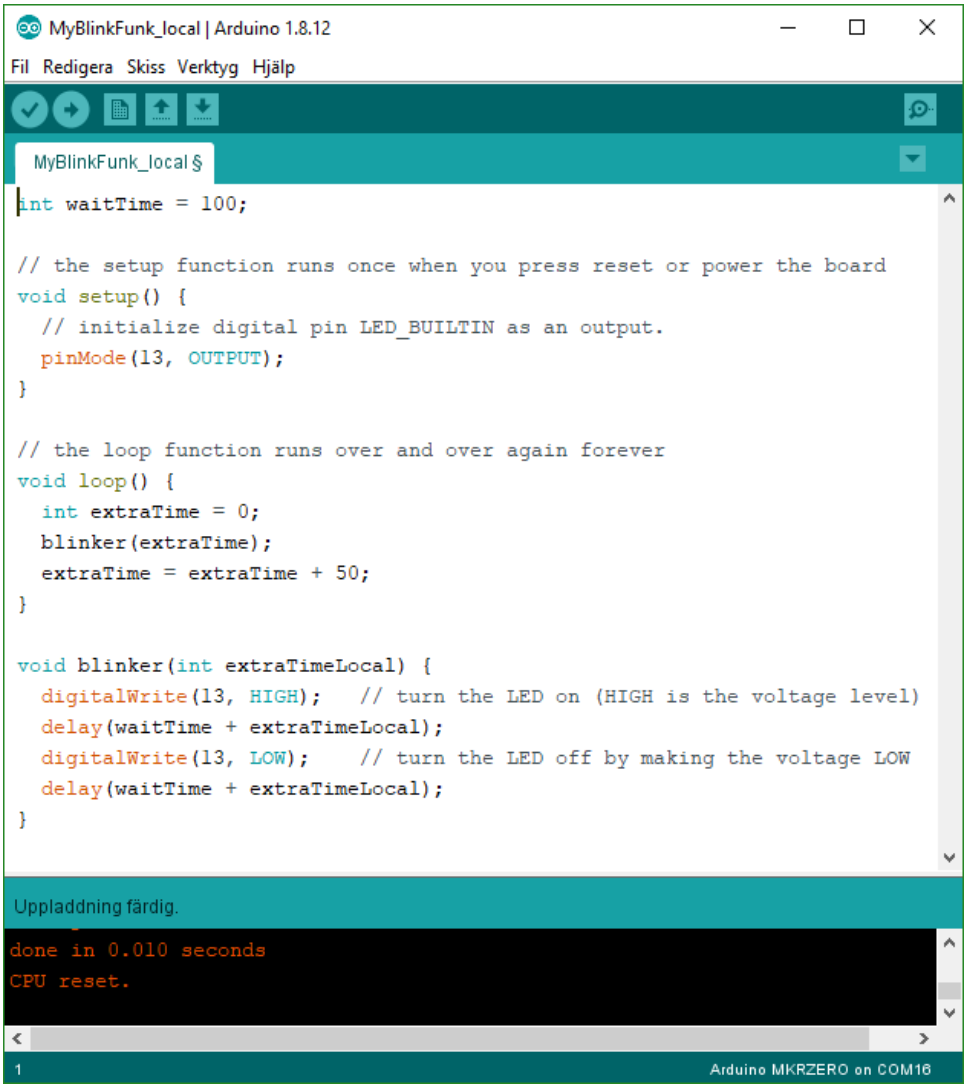
Nu ska vi titta på ett annat sätt att lösa uppgiften. Flytta deklarationen av `extraTime` in i `loop`. Detta gör att den variabeln inte finns utanför `loop` längre. Vi kan dock skicka den till `blinker` där den behövs. Det gör vi genom att skriva namnet på variabeln vi skickar i parentesen där vi anropar funktionen.

- `blinker(extraTime);`

Vi behöver också ändra den mottagande änden genom att definiera vilken typ av variabel funktionen ska vänta sig. Detta gör vi genom att skriva typ och namn i parentesen efter deklarationen av funktionen. Notera att detta är en definition av en ny variabel så typ behöver anges. Detta innebär även att vi kan använda ett nytt namn. Vi borde inte använda `extraTime` trots att den är ledigt och skulle fungera, för att motverka förvirring. Vi kan kalla den nya variabeln `extraTimeLocal`. Skriv alltså:

- `void blinker(int extraTimeLocal)`

Vi behöver också byta namn på variabeln i blinker-funktionen så de stämmer med variabelns lokala namn. Byt namn på `extraTime` i `blinker` till `extraTimeLocal`.



Så som koden är skriven nu kommer den lokala variabeln omdefinieras varje loop innan beräkningarna sker. Med andra ord så kommer tiden mellan pulserna inte öka trots att det är det vi vill. Nyckelordet `static` före en variabeldeklaration gör att en definition endast utförs om variabeln inte redan finns. Lägg till `static` innan `int`.

- `static int extraTime` = 0;

Pulserande diod

För att utveckla vårt system ska vi koppla en diod som ändrar ljusstyrka. Först, koppla en lysdiod som ni har gjort tidigare, i serie med en resistor, med minus kopplad mot jord och plus mot en pin.

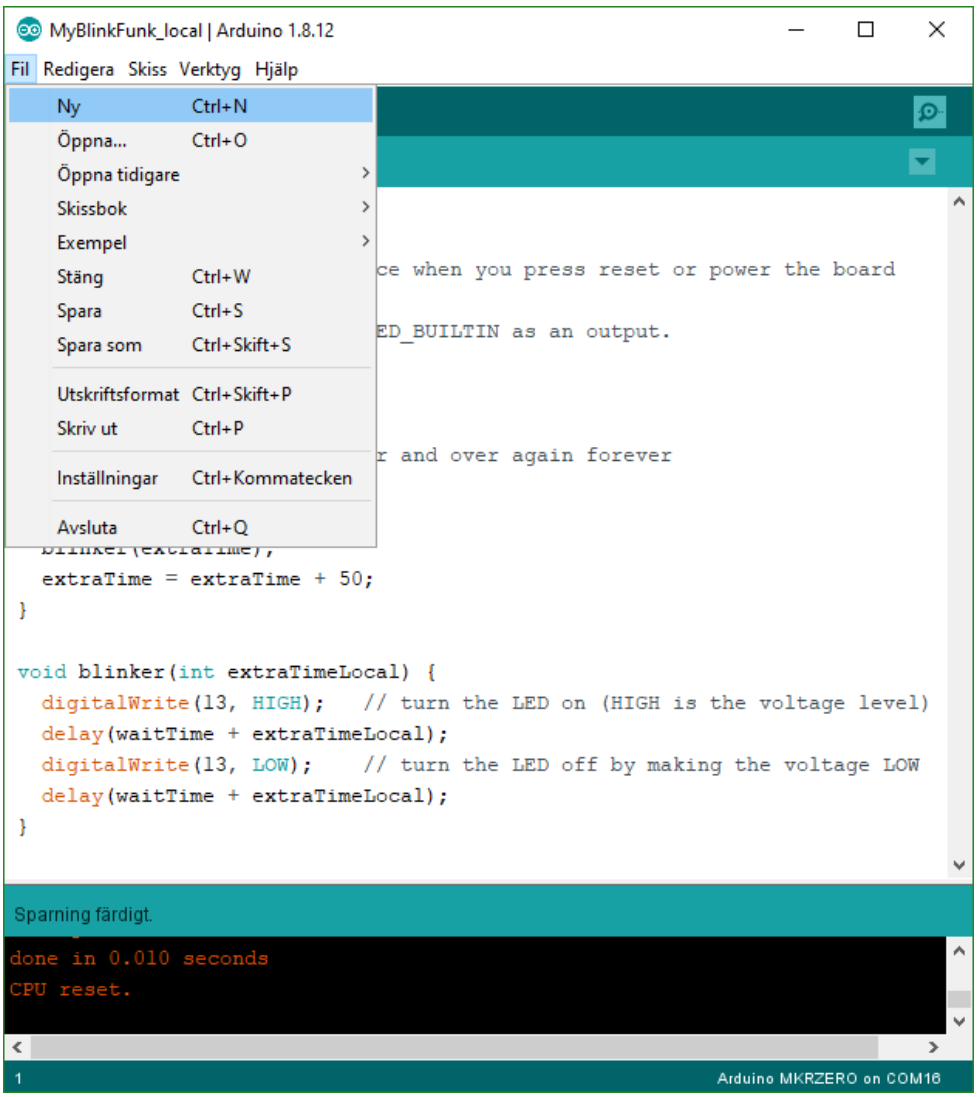
Att göra: byt kopplingen från den tidigare pin mot *pin 10.*

Skissen vi kommer skriva kommer fungera på följande sätt. Först bestäms en pin som en lysdiod är kopplad på. I det här fallet är det pin 10. Den pin man väljer måste vara kapabel att reglera spänningen exakt, (så kallat Pulse Width Modulation). På den länkade sidan i dokumentationen finns en tabell över vilka pins som har den kapaciteten för varje typ av mikrokontroller[3].

Om man kopplar på en pin som saknar denna egenskap så kommer dioden inte ändra ljusstyrka utan hålla en konstant, låg, ljusstyrka. I varje steg av loopen ska vi öka eller minska ljusstyrkan på dioden med ett steg och vi vill vända när vi har nått till maximal eller minimal styrka.

Att göra: Öppna en ny skiss. Det gör du under Fil -> Ny.

[3] https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/?from=AnalogWrite.PWM



Skapa en variabel för att hålla numret på den pin lysdioden är kopplad på.

- `int ledPin = 10;`

I `setup`, öppna den pin-en med med `pinMode()`

- `pinMode(ledPin, OUTPUT);`

`pinMode()` tar en pin (en siffra) och ett läge den ska sättas i. I vår `loop` vill vi göra två saker. Vi kommer använda den inbyggda funktionen `analogWrite()` istället för `digitalWrite()` eftersom det kommer låta oss ha fler olika värden på vår utsignal (lysdiodens ljusstyrka) än bara HIGH och LOW och sen vill vi kolla om ljusstyrkan är i något av ändlägena och i så fall ändra hur ljusstyrkan ändrar sig. `analogWrite()` tillåter värden mellan 0 och 255.

Definiera en statisk lokal variabel som kommer vara ljusstyrkan (`k`) och en som är förändringen (`delta`) i varje varv av loopen.

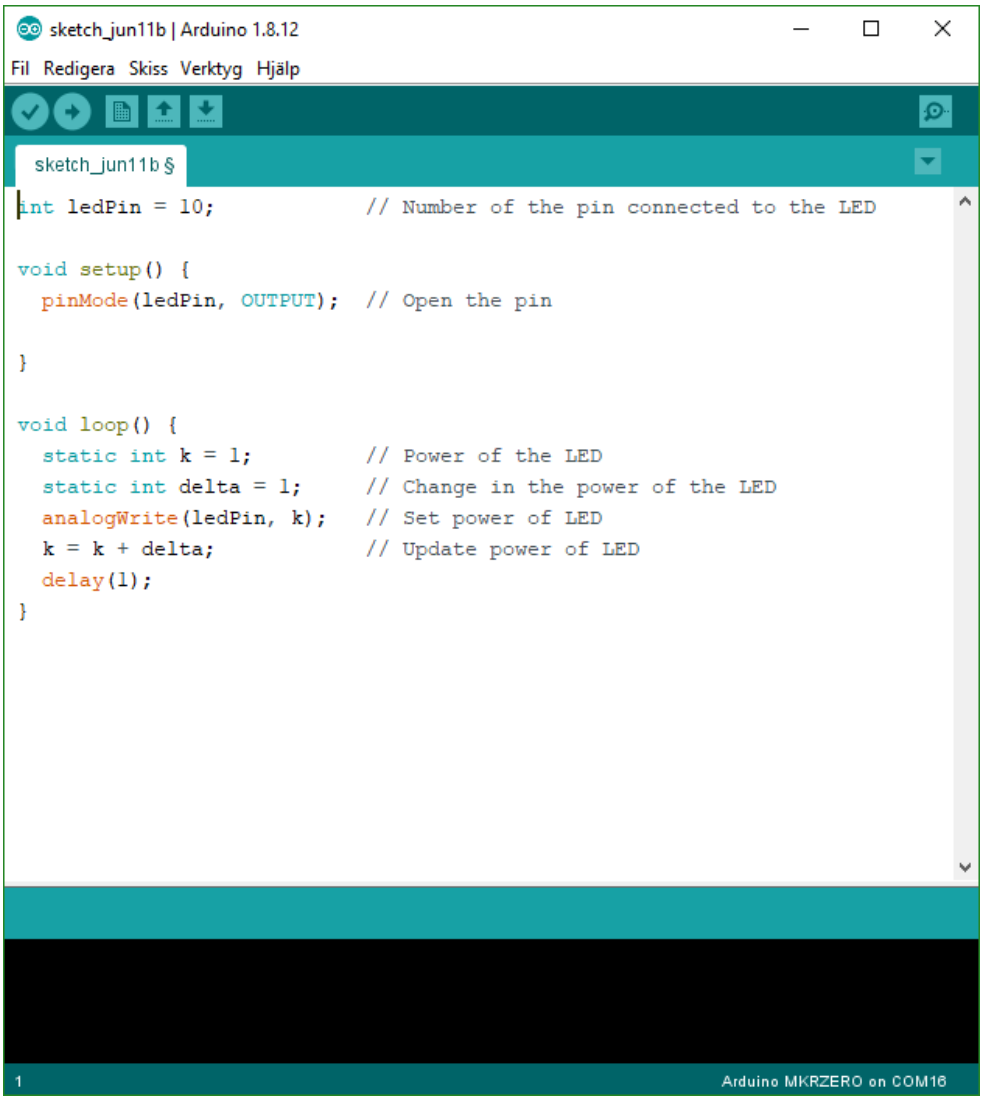
- `static int k = 0;`
- `static int delta = 1;`

Sedan skickar det till dioden med `analogWrite()` och uppdaterar ljusstyrkan genom att skriva ljusstyrka = ljusstyrka + förändring.

- `analogWrite(ledPin, k);`
- `k = k + delta;`

En kort `delay` behövs för att processen ska gå långsamt nog att se.

- `delay(1);`



Problemet nu är att ljusstyrkan kommer komma till 255 och sedan bara fortsätta räkna. Vi vill att den ska vända och gå tillbaka mot noll. Det kan vi göra med en så kallad if-sats. En if-sats är en bit kod som gör något om ett visst kriterium uppfylls. Vi vill att vår förändringsvariabel blir negativ om ljusstyrkan är maximal. I C skrivs en if-sats så här

```
    if( kriterium )
{
kod
}
```

där kriteriet måste vara ett booleskt uttryck dvs något som är sant eller falskt.

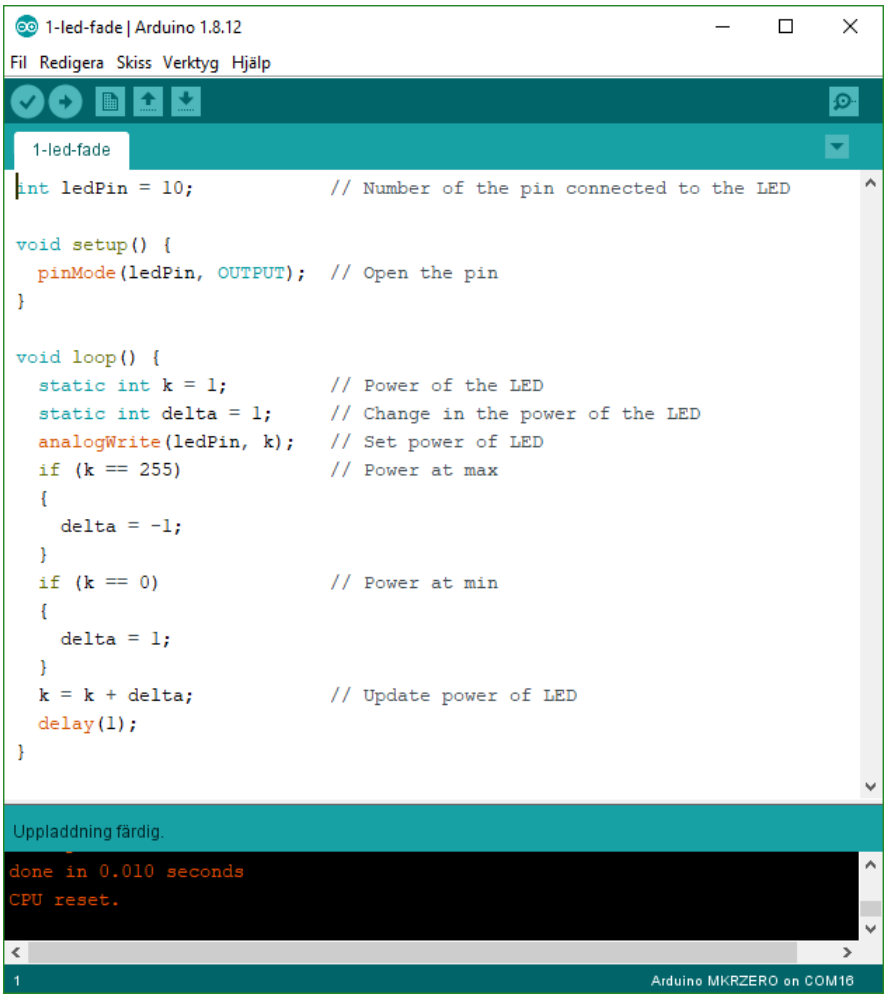
Det finns sex stycken boolska operatorer som visas i tabellen nedan.

Operator	Betydelse	Exempel	Resultat
<	Mindre än	1 < 2 2 < 2	Sant Falskt
>	Större än	2 > 1 2 > 2	Sant Falskt
<=	Mindre eller lika med	1 <= 2 2 <= 2	Sant Sant
>=	Större eller lika med	1 >= 2 2 >= 2	Falskt Sant
==	Lika med	2 == 2	Sant
!=	Inte lika med	2 != 2	Falskt

Vi vill ha två if-satser, en för när vi har räknat till 255 och en för när vi kommit tillbaka till 0. Vi skriver våra if-satser så här:

- ```
  if (i == 255)
  {
  delta = -1;
  }
  ```
- ```
  if (i == 0)
  {
  delta = 1;
  }
  ```

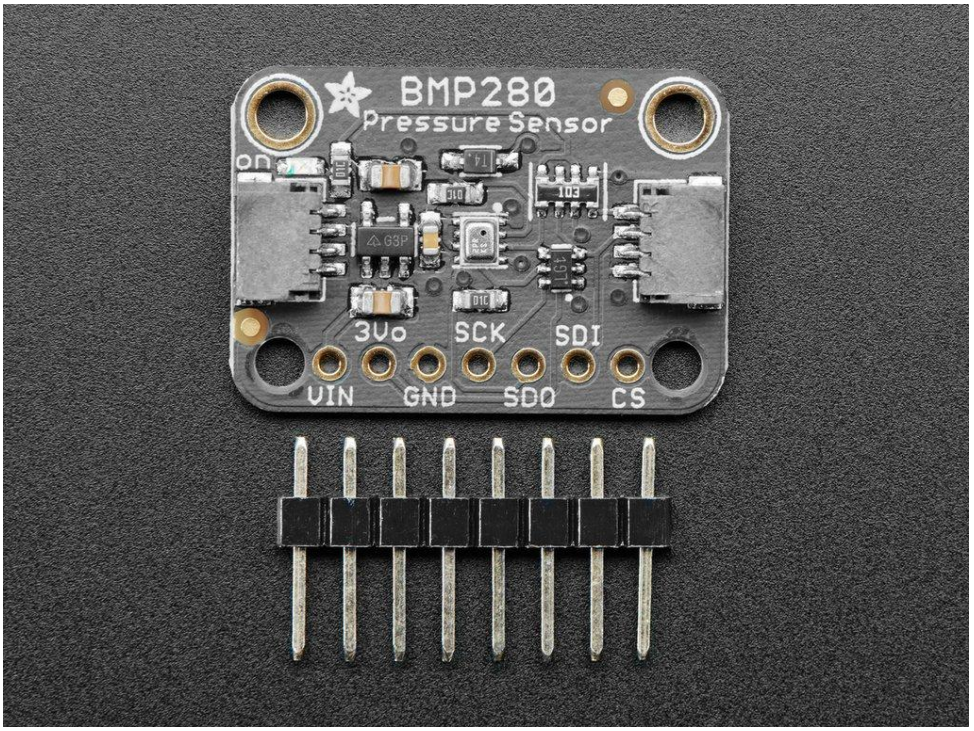
I varje varv av loopen kommer programmet nu att kolla om något av kriterierna är sanna. Om något är det kommer den koden att köras, alltså kommer `delta` ändra värde.



Prova programmet genom att ladda upp den till Arduinon. Vad händer med dioden?

Temperatur- och lufttrycksmätare

Vi kommer nu gå över till att använda en sensor som kan mäta temperatur, lufttryck och även höjd över havet. Sensorn heter BMP280 och går att använda i en CanSat tävling för att utföra det primära uppdraget.



Sensorn ska kopplas med en pin till +5V, en till jord och sedan kan man välja på två kopplingsscheman. Man kan välj på att använda en så kallad I2C koppling. Då behöver man endast använda 4 sladdar.

Om man använder I2C kopplar man enligt följande tabell:

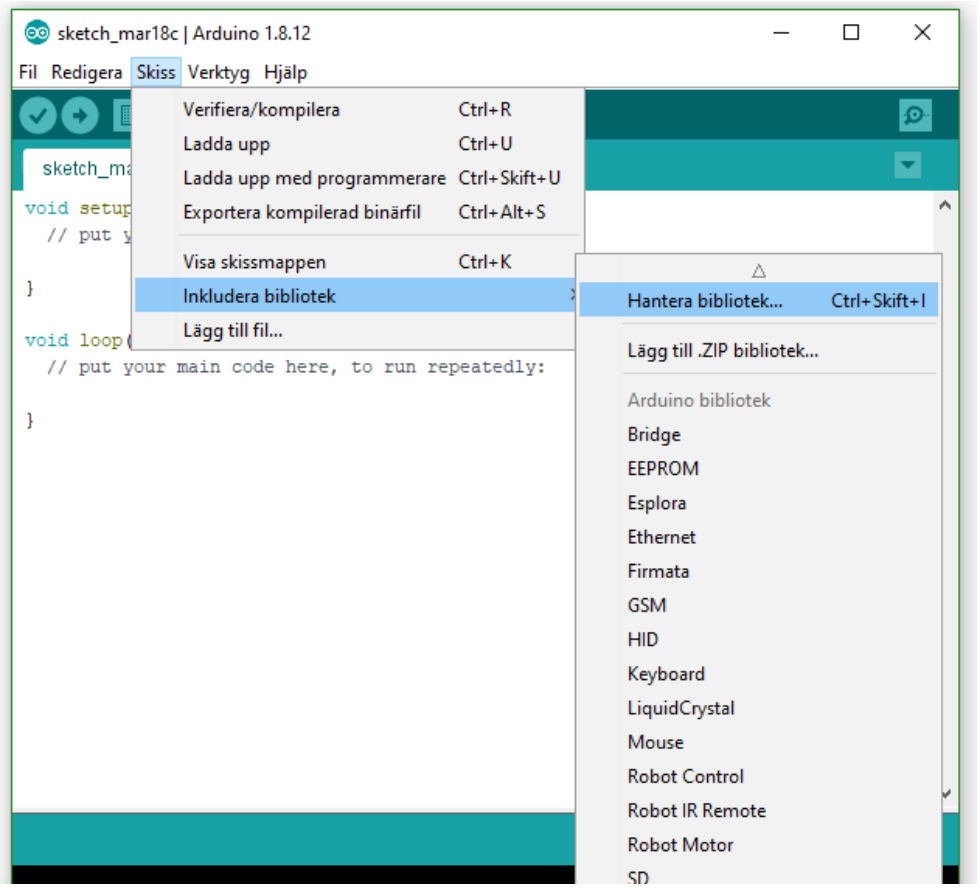
Pin på kort	Pin på Arduino
Vin	+5V
GND	GND

SCK	SCL
SDI	SDA

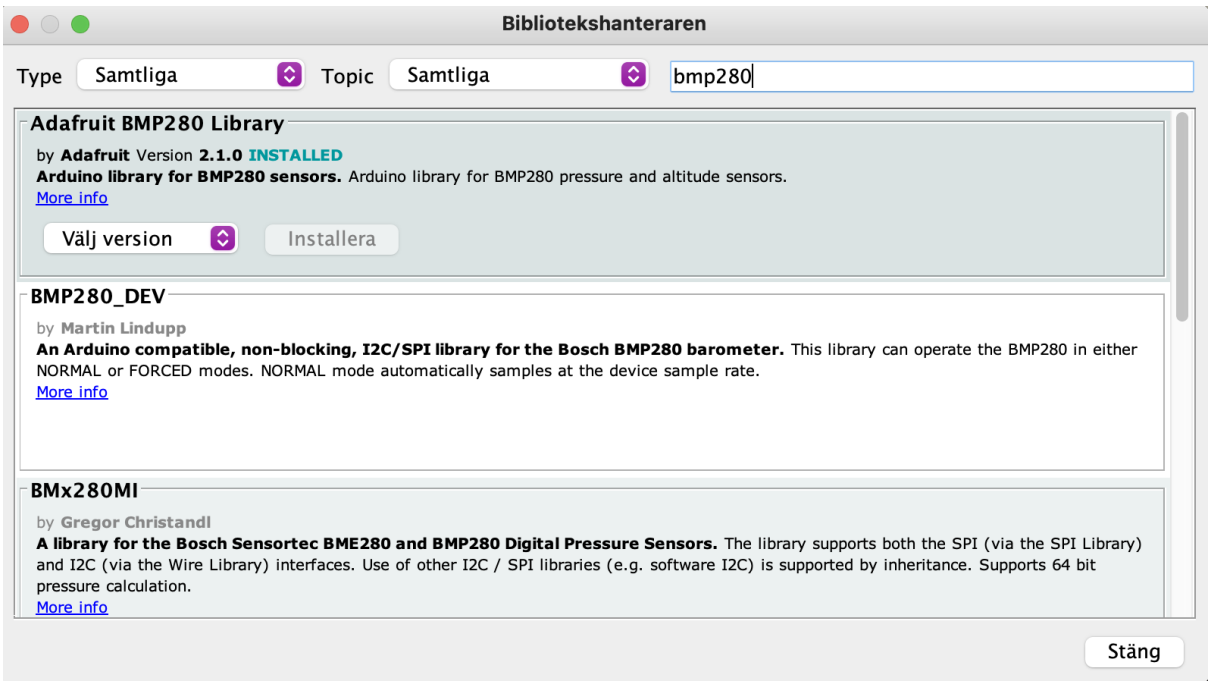
Man kan också använda SPI koppling som kräver fler kopplingar:

Pin på kort	**Pin på Arduino**
Vin	+5V
GND	GND
SDI	MOSI / 8
SDO	MISO / 10
SCK	SCK / 9
CS	13

Ni kan prova båda kopplingarna och se vad som passar er bäst. Först måste vi bara ladda ner rätt mjukvara så att vi kan läsa av mätdatan från sensorn. Den hittar man genom att gå in i Skiss -> Inkudera bibliotek -> Hantera bibliotek



Skriv bmp280 i sökrutan i fönstret. Välj "**Adafruit BMP280 Library**" och tryck på installera. Programmet kommer fråga om det ska installera andra bibliotek som Adafruit BMP280 Library beror på, välj install all.



I2C

Öppna en ny skiss. För att använda det nya biblioteket skriver du kommandot:

- `#include "Adafruit_BMP280.h"`

Detta tar in koden från det nedladdade biblioteket i skissen. Notera att det inte ska vara något semikolon efter denna rad. Programmet kommer ge ett konstigt felmeddelande om det finns ett semikolon efter `#include` eller `#define`. Vi kommer använda digitala pins så vi behöver inte definiera vilka pins som används, men vi behöver tala om vilken sensor vi använder. Vi skriver

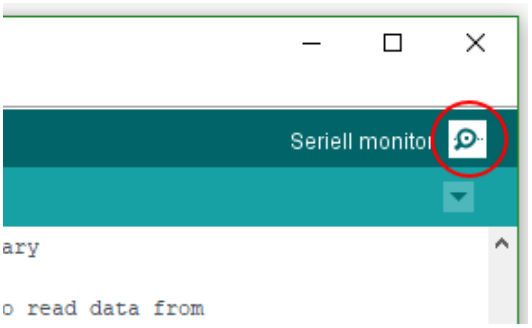
- `Adafruit_BMP280 bmp;`

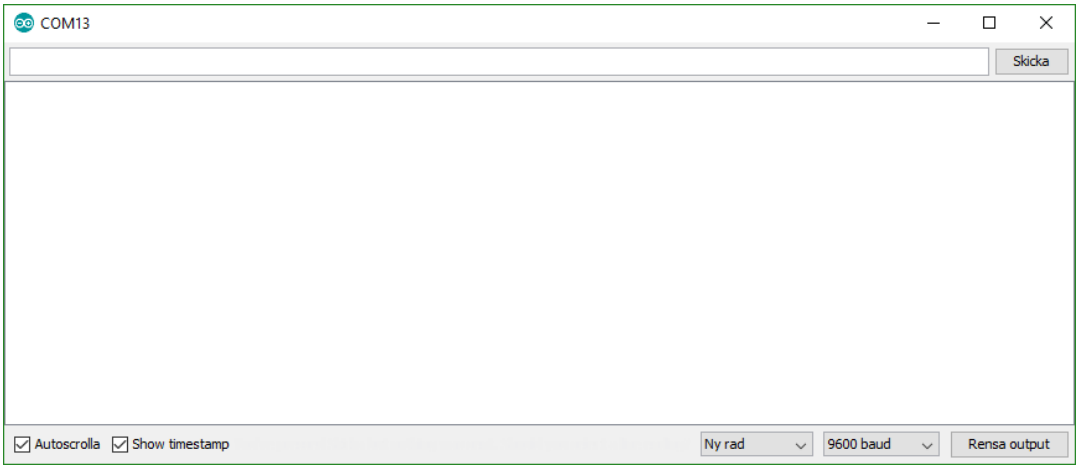
för att etablera ett `Adafruit_BMP280` objekt som kallas `bmp`. Till sist behöver vi skriva

- `bmp.begin();`

i `setup` för att starta koden. Den kod som är skriven hittills kommer inte göra något, den är endast det som behövs för att kunna börja.

För att kunna se mätvärdena använder vi Arduinos inbyggda serial monitor. Det är ett fönster som visar data som kommer tillbaka till programmet från arduinon. Man öppnar fönstret genom att klicka på knappen längst upp till höger.





För att starta utskriften skriver vi

- `Serial.begin(9600);`

i `setup`. Siffran är en hastighetsparameter, punkten innebär att `begin` är en funktion inom `Serial`. För att skriva ett meddelande till serial monitor använder man `Serial.print("meddelande")`. Om man vill skriva vanlig text måste den vara inom citationstecken, om man vill skriva värdet av en variabel ska den vara utan citationstecken. Man kan t.ex. skriva :

- `Serial.println("Start of BMP280 measurements");`

i setup. För att byta rad använder man `Serial.println`, den funktionen går till nästa rad efter att den har skrivit sitt meddelande.

I `loop` behöver vi läsa in värden från sensorn och ge det till ett par variabler. Vi ger de nya variablerna typen `float`, inte `int`, eftersom vår data utgörs decimaltal (floating point number). Vi skriver

- `float t = bmp.readTemperature();`
- `float p = bmp.readPressure();`
- `float h = bmp.readAltitude(1012.3);`

Dessa kommandon använder funktioner i biblioteket vi la till för att läsa av värdet i sensorn och spara dom i variablerna vi anger. Temperaturen anges i grader Celsius, lufttrycket anges i pascal och höjden/altituden anges i meter. För att höjden ska beräknas korrekt behöver man ange lufttrycket vid havsytan, 1012.3 hPa för Stockholm enligt SMHI[4]. För att skriva resultaten till serial monitor skriver man:

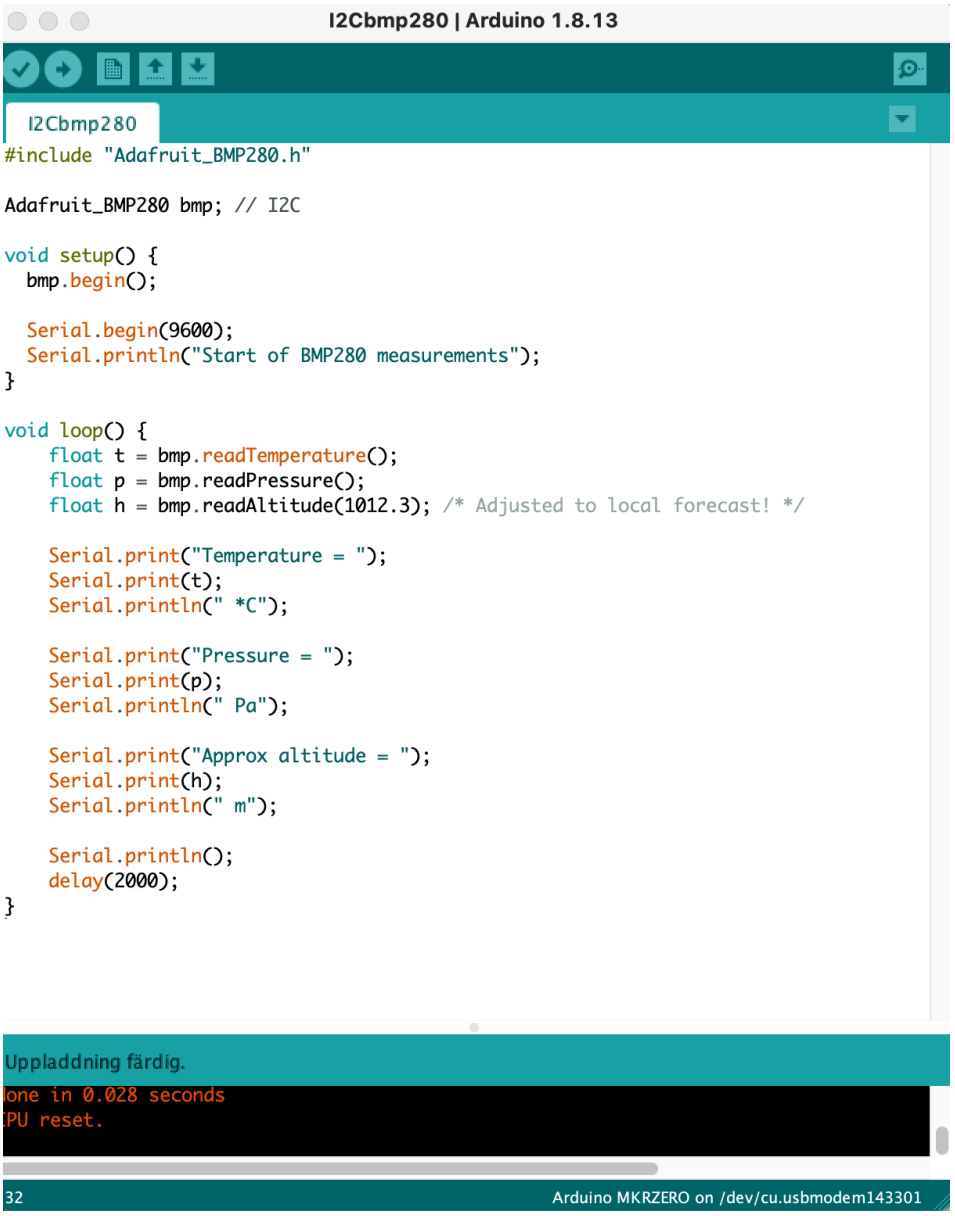
- `Serial.print("Temperature = ");`
- `Serial.print(t);`
- `Serial.print(" *C");`
- `Serial.print("Pressure = ");`
- `Serial.print(p);`
- `Serial.println(" Pa");`
- `Serial.print("Approx altitude = ");`
- `Serial.print(h);`
- `Serial.println(" m");`

Lägg slutligen in en

- `delay(2000);`

för att göra en mätning varannan sekund.

[4] https://www.smhi.se/kunskapsbanken/meteorologi/lufttryck-1.657



Om koden körs nu kommer den skriva ut mätvärden i monitorn.

Att göra: Verifiera koden och testa att köra den. Är värdena rimliga?

Fungerar det inte? Ett vanligt fel är att istället för siffror skriver koden ut `nan` (not a number). Detta beror oftast på att sensorn är felkopplad. Dubbelkolla att sensorn har så bra kontakt med sina pins som det går. För att få bättre kontakt mellan sensorn och pinsen kan man löda fast dem. Fråga din lärare om du vill göra det.

SPI

Vi ska nu titta på SPI koppling som är analog, vilket betyder att vi måste definiera de pins som används. Det enda som behöver ändras i programmet ovan är att

- `Adafruit_BMP280 bmp;`

behöver bytas ut till

- `Adafruit_BMP280 bmp(BMP_CS, BMP_MOSI, BMP_MISO,  BMP_SCK);`

och ovanför definierar vi pinsen som används genom att skriva

- `#define BMP_SCK  9`
- `#define BMP_MISO 10`
- `#define BMP_MOSI 8`
- `#define BMP_CS   13`

Sedan är det bara att köra programmet igen och läsa av mätvärdena som ovan.

Extra övning

Koppla en krets och skriv en skiss som tänder en lysdiod när man trycker ner en knapp. För att göra det svårt ska dioden inte vara kopplad tillsammans med knappen, dvs när knappen trycks ner får Arduinon en signal som får den att tända dioden.