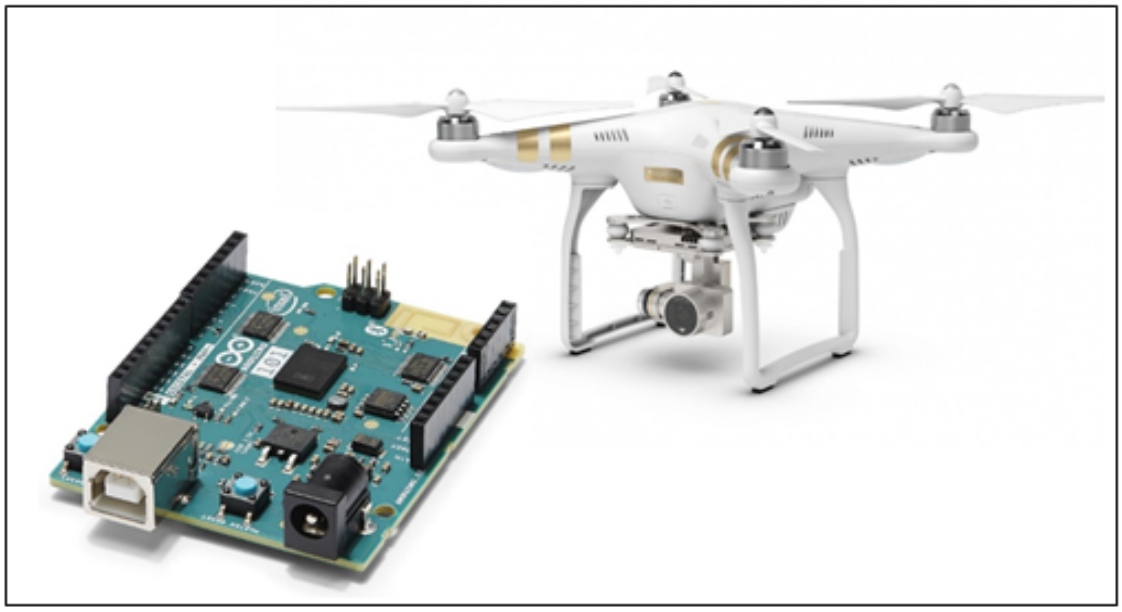
# Mijn eerste mini-satelliet

**Wie een elektronisch experiment leert bouwen, opent mogelijkheden voor talloze STEM uitdagingen op school, en kan deelnemen aan schoolprojecten van ESERO en ESA. We maken een eerste eenvoudig satellietje die de temperatuur en luchtdruk kan meten. Vervolgens 'lanceren' we het met een drone. Voor absolute beginners.**



Belgium esero

FAB LAB KLEIN BRABANT

**Uitgave** November 2017

**Doelgroep**

Basis K 1 2 3 4 5 6

Secundair 1 2 3 4 5 6

# Mijn eerste mini-satelliet

## Metingen tijdens een vlucht – voor absolute Arduino beginners

# Cursus kenmerken

**Leeftijd doelgroep** 14-18 jaar

**Type** Informatieve cursus met oefeningen en instructies

**Vereist gebruik van**

- Arduino UNO microconrtoller en accessoires
- Luchtdruksensor en temperatuursensor
- Soldeerstation
- LED's en weerstanden
- Micro SD kaart en datalog shield

**Eindtermen** Diverse STEM eindtermen
(ideaal voor projectonderwijs, vrije ruimte, ...)

**Samenvatting** Hoe kan je metingen uitvoeren op een onbemande vlucht (drones, miniatuuvliegtuigen, weerballon, raket, ..)? Hiervoor heb je basis elektronica nodig. In deze cursus leren absolute beginners om de luchtdruk en de temperatuur te meten via een Arduino microcontroller. Je leert eenvoudige basisvaardigheden zoals solderen, bedienen van een LED, communiceren met de microcontroller (via laptop), aanpassen van de programmeercode, en opslaan van gemeten data.
Deze vaardigheden vormen een basis om deel te nemen aan projecten zoals ASGARD, CanSat of Bifrost. Hierin worden leerlingen uitgedaagd om een zelfbedacht experiment te lanceren.

# Colofon

**Uitgave** November 2017

**Laatste update** 9 nov 2017



**AUTEURS**

**ESERO Belgium**

- Inhoud van de cursus en layout
- Vormingen voor leraren: organisatie
- Medewerker: Pieter Mestdagh

**FabLab Klein-Brabant**

- Aanvullingen
- Lesgever van de vormingen
- Medewerker: Davy Van den Bergh

**Uw mening is belangrijk**

Cursussen van ESERO Belgium worden online aangeboden in dynamische vorm. Dat betekent dat elke zinvolle feedback van gebruikers leidt tot de publicatie van een aangepaste uitgave op www.esero.be (Nederlandstalig). Help toekomstige gebruikers door uw opmerkingen of aanvullingen per email op te sturen (www.esero.be > NL > contact).
Gebruikers die nieuwe onderdelen toevoegen aan de cursus, worden hierboven in de auteurslijst vermeld.

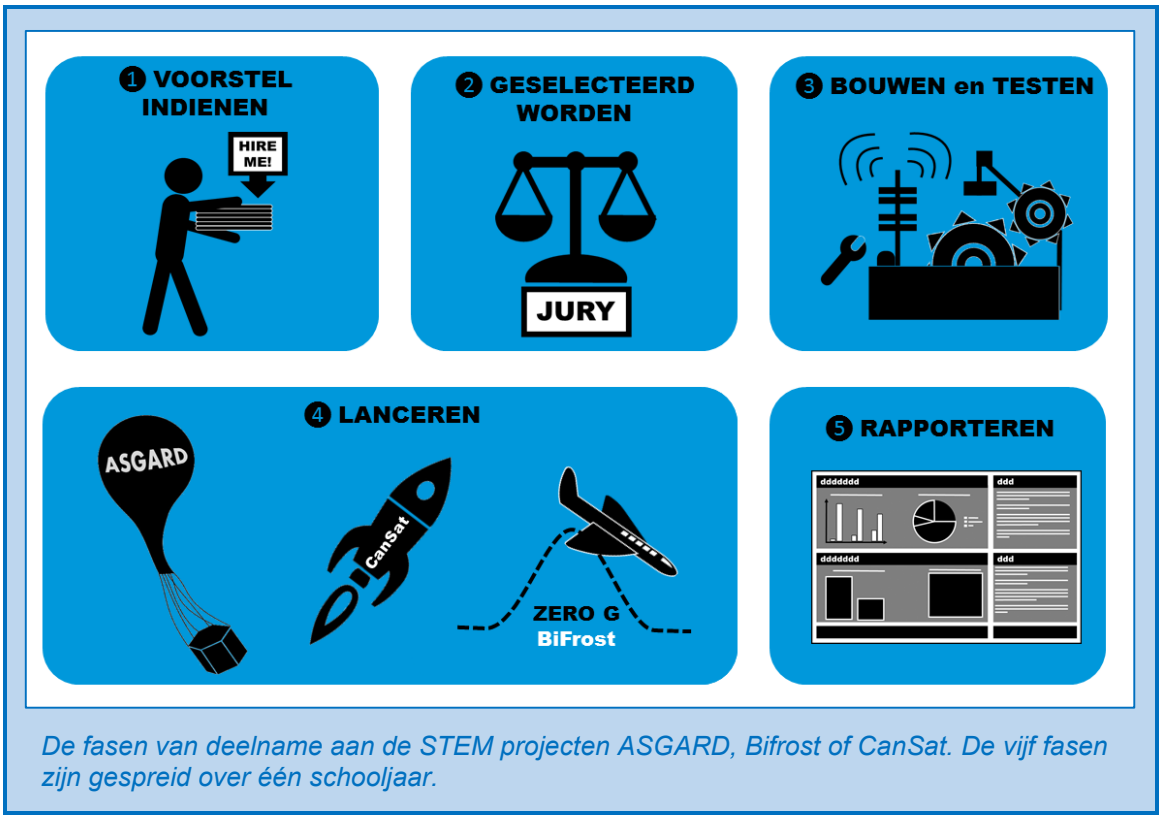
# Inhoud

# 1 Bouw je eerste satelliet

## 1a A space mission at school:

Elk jaar kunnen Belgische en buitenlandse secundaire scholen deelnemen aan STEM projecten waarbij leerlingen hun eigen experiment lanceren in een educatieve “ruimtemissie”. Het patroon is bij elk project hetzelfde:

1) **Inschrijving**: De leerlingen bedenken zelf een experiment, en beschrijven hun idee in een dossiertje. Een leraar wordt aangeduid als begeleider van dit team.
2) **Selectie**: een jury van deskundigen selecteert zoveel mogelijk inschrijvingen. De jury is niet noodzakelijk op zoek naar de ‘beste’ inschrijvingen. Vooral de leerervaring is belangrijk.
3) **Het experiment bouwen**: Eens geselecteerd gaan de leerlingen hun zelfbedacht experiment in elkaar knutselen en testen. Een moeilijke, maar leerrijke fase.
4) **Lancering**: Afhankelijk van het project wordt het experiment gelanceerd met een stratosfeerballon, een kleine raket, of in een paraboolvlucht in gewichtloosheid.
5) **Rapportering**: Zowel tijdens als na het projectverloop moeten de teams rapporteren over hun werk, los van he al dan niet bekomen van zinvol resultaat.




*De fasen van deelname aan de STEM projecten ASGARD, Bifrost of CanSat. De vijf fasen zijn gespreid over één schooljaar.*

## Drie verschillende “space missions”

De tabel hieronder geeft een overzicht van de drie STEM projecten die aangeboden worden aan Belgische scholen. Voor meer informatie: ga naar www.esero.be (projecten).

| Project | Beschrijving | Wanneer | Teams |
|---|---|---|---|
| **ASGARD** | Je experiment bereikt met een stratosfeerballon een hoogte van +/- 30 km. Dat is boven de Ozonlaag, de omstandigheden zijn vergelijkbaar met het oppervlak van Mars. | Elk schooljaar (inschrijven voor 11 nov) | • Max 5 leerlingen<br>• 12-20 jaar<br>• 1 leraar (max 2)<br>• Voor scholen uit alle landen |
| **Bifrost** | Je monteert je experiment in een klein vliegtuig, waarin je zelf ook meevliegt. Er worden meerdere parabolen gevlogen waarin je telkens +/- 20 sec gewichtloos bent. | Om de twee jaar (volgende editie: 2018-2019) | • 16-20 jaar<br>• 1 leraar<br>• Voor Belgische scholen |
| **CanSat Belgium** | Je lanceert je satelliet ter grootte van een 33cl blikje met een kleine raket naar een hoogte van 1 à 2 km. Je recupereert de satelliet via een veilig landingssysteem, en houdt radiocontact tijdens de vlucht. | Om de twee jaar (volgende editie: 2017-2018) | • 18 teams (12 lanceringen)<br>• 4 tot 6 leerlingen<br>• 14-20 jaar<br>• 1 leraar<br>• Voor Belgische scholen |
| **CanSat Europe** | De nationale winnaars komen ergens in Europa samen om hun CanSat te lanceren in een Europese competitie. | Einde van elk schooljaar (Belgische deelname om de twee jaar) | • 1 team per land<br>• 4 tot 6 leerlingen<br>• 14-20 jaar<br>• 1 leraar |


*Kort overzicht van de drie STEM projecten waarin een “space mission” wordt gesimuleerd. De drie Belgische space-projecten worden ook afgekort als de ABC-projecten vanwege hun beginletters.*

### Weetjes

- ASGARD is na de jury-selectie geen competitie, en staat open voor scholen uit de hele wereld.
- CanSat vindt ook plaats in andere Europese landen, en er is jaarlijks een Europese competitie (georganiseerd door ESA) waaraan per land 1 nationaal winnend team kan deelnemen.
- ASGARD en CanSat hebben Engels als voertaal.
- Bifrost is uitsluitend open voor Belgische scholen. De teams kunnen in dat project rapporteren en communiceren in de taal van hun school (Nederlands, Frans, Duits of Engels).
- Op de ASGARD vlucht worden jaarlijks aanvullend twee klassen uit derde graad Belgisch lager onderwijs toegelaten.

### Elektronische experimentjes voor absolute beginners

Hoewel er in de projecten ASGARD en Bifrost voorbeelden zijn van niet-elektronische experimenten, gebruiken de meeste leerlingenteams een eenvoudige microcontroller om hun experiment aan te sturen.

ESERO Belgium biedt aan Belgische leraren een vorming aan waarin men een eerste ervaring opdoet met het gebruik van de microcontroller Arduino UNO.

### Maak je eerste mini-satelliet

In deze cursus gaat men onder begeleiding een eerste ‘satelliet’ maken die de luchtdruk en de temperatuur kan meten. Op het einde van de vorming maakt de satelliet van elke deelnemende leraar een vlucht met een drone. Vervolgens kan men de temperatuur, luchtdruk en hoogte van de vlucht uitzetten op een grafiek. De vorming haalt absolute beginners over de drempel om een elektronisch aangestuurd experiment te bouwen met een leerlingenteam, en aldus in te schrijven voor de STEM projecten ASGARD, Bifrost, en CanSat.

Voorliggende cursus is de (uitgebreide) neergeschreven versie van de lerarenvorming “Mijn eerste mini-satelliet”.
Uiteraard kan deze cursus ook een goede inleiding zijn om een µcontroller te gebruiken voor andere STEM projecten op school.

## 1b Onderdelen van een satelliet (payload, bus)

### U maakt een satelliet

Strikt genomen maakt u in deze vorming geen satelliet. Een satelliet is immers een kleiner object dat in een baan om een groter hemellichaam cirkelt. We onderscheiden natuurlijke satellieten (zoals een maan) en kunstmatige satellieten. Deze laatste zijn door mensen gemaakte apparaten die in een baan rond de Aarde of een ander lichaam bewegen. We noemen het apparaat dat u zal vliegen toch een satelliet omwille van volgende gelijkenissen:

- Wat u maakt is de “**payload**”, en de drone waarmee het vliegt is de “**bus**”. Deze termen worden verder uitgelegd.
- Uw **metingen** dienen volledig **autonoom** uitgevoerd te worden eens uw payload in de vlucht is. Alles wordt vooraf op de grond geprogrammeerd en voorbereid.
- De satelliet werkt autonoom, maar wordt bijgestaan door de ‘**ground control**”. In dit geval is dat de besturing van de drone (door de drone piloot) en de dataverwerking van de metingen op uw laptop. Indien u de data live tijdens de vlucht via radiocommunicatie ontvangt, dan is ook de grondantenne deel van de “ground control”.

## Payload

De payload is het gedeelte van de satelliet waarmee metingen of experimenten uitgevoerd worden. De payload is als het ware de nuttige lading. Dit laatste is trouwens de Nederlandse vertaling van het Engelse woord payload.

In deze vorming bestaat onze payload uit volgende onderdelen:
- de micorcontroller (Arduino UNO)
- sensoren voor druk en temperatuur
- de geheugenkaart of de data-transmitter
- verbindingskabels
- weerstanden en LEDs

Ze worden één voor één besproken in onderstaande hoofdstukken.

## Bus

Een **bus** is in de elektronica een medium waarop verschillende elektronische signalen worden uitgewisseld. Het is dus een netwerk waarop meerdere componenten of apparaten met elkaar in contact staan.

Bij een satelliet is de **bus** is het geheel van subsystemen die de satelliet correct laten functioneren en in zijn baan laat vliegen. In onderstaande lijst worden algemene subsystemen van satellieten vergeleken met de subsystemen van onze drone-satelliet.

| Algemene sateliet bus onderdelen | Onze mini-satelliet |
|---|---|
| De structuur van het ruimtetuig | De drone en het harnas waarin de payload wordt vastgezet |
| Communicatiemechanismen om de satelliet te laten communiceren met de grondcontrole | Communicatiemechanismen die de drone laten communiceren met de piloot interface |
| Navigatiesystemen en telemetrie | Hoogtemeter en gps van de drone |
| Motoren en brandstofcellen | Motoren en batterijen van de drone |
| Temperatuur controle systemen | Niet van toepassing |
| Energievoorziening (via brandstof of zonnepanelen) | Batterij (power bank) die we meesturen met onze payload |

# 2 Payload

Alle componenten die u nodig heeft om de payload voor deze vorming te maken worden hieronder uitgebreid beschreven. Neem dit gedeelte eerst door, zodat u de componenten goed kent, en begrijpt hoe ze werken. In een volgende stap kunnen ze dan ingeschakeld worden in een circuit met de Arduino UNO.

**Lijst van componenten die u meekrijgt wanneer u deze ESERO vorming volgt:**

| Categorie | Onderdelen |
|---|---|
| Microcontroller | • Arduino UNO board |
| Sensoren | • TMP36: temperatuursensor<br>• BMP280: sensor voor druk, temperatuur en hoogte |
| Hulpstukken | • USB kabel A/B<br>• Gekleurde Jumper kabeltjes<br>• Weerstanden 10 K en 100 Ohm<br>• LEDs R470 groen, oranje, rood<br>• Taktiele schakelaar 12 mm<br>• Breadboard medium afmeting<br>• Stacking headers voor Arduino R3 |
| Data opslag | • DataLog Shield met RTC 1307<br>• Micro SD kaart 16 GB |

**Stroomvoorziening**

Tijdens de vorming verbindt u de Arduino UNO met uw laptop. Op dat moment wordt de microcontroller van stroom voorzien via de USB verbinding.
Tijdens de vlucht zorgt ESERO BE voor een externe batterij waarop u uw Arduino kan aansluiten. Een batterij of power bank wordt niet meegeleverd aan de cursisten.

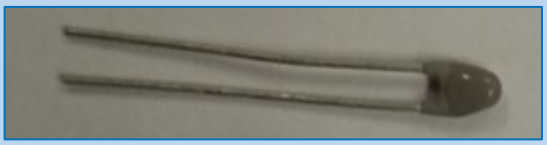
## 2a Perifere onderdelen van de payload

### TEMPERATUUR SENSOR

Hoewel de vorming één bepaalde temperatuur sensor gebruikt (TMP36), is het interessant om ook een andere veel voorkomende soort temperatuursensor te leren kennen: de NTC

#### De NTC thermistor

NTC staat voor ”negative temperature coefficient”. De naam is logisch: wanneer de temperatuur stijgt, dan zal de elektrische weerstand van de thermistor dalen, en omgekeerd. Het gevolg is dat de Arduino een veranderde spanning zal registreren over deze sensor. Deze veranderde spanning kan omgerekend worden naar veranderde weerstand, en vervolgens naar veranderde temperatuur. De NTC thermistor is erg eenvoudig, maakt geen interne bewerkingen of berekeningen, en heeft maar twee ‘pootjes’, net als een gewone weerstand.



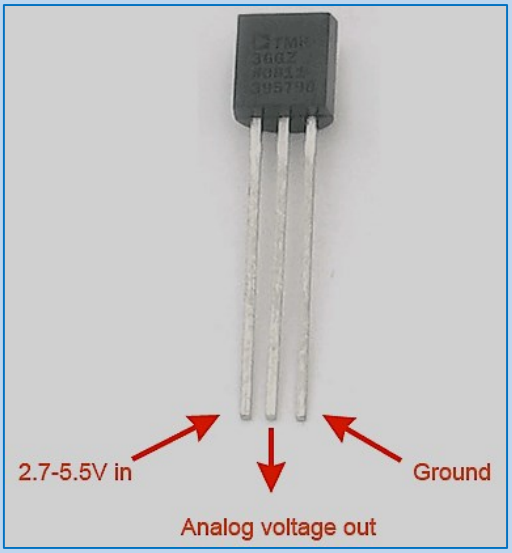
*NTC thermistor. Bron: ESA/ESERO werkgroep Space Robotics.*

Het woord **thermistor** is een samenvoeging van de Engelse woorden ‘thermo’ en ‘resistor’ (weerstand). De NTC is namelijk een electrische weerstand waarvan de waarde afhankelijk is van de temperatuur.

## De TMP36 sensor: introductie

Tegenwoordig bevatten de meeste sensoren meer dan alleen maar de actieve component die verantwoordelijk is voor de variabele output (zoals de weerstand van de NTC). Met behulp van een set transistoren voert de sensor reeds bewerkingen uit, zodat de output beter overeenkomt met de gewenste variabele.

Dit is ook het geval met de TMP temperatuursensor.




*De TMP36 sensor. Bron: Adafruit.*

## De TMP36 sensor: principe, werking

Wanneer de temperatuur stijgt, dan daalt de spanning over een diode met bekende hoeveelheden. Dit principe wordt gebruikt om de variatie in temperatuur te registeren in de TMP sensor. Men gebruikt echter geen diode, maar een transistor. Wanneer de temperatuur varieert, dan varieert de spanning over de basis en de emitter van zulke transistor. Deze spanningsvariaties worden verder met versterkt om een meting mogelijk te maken.

De berekeningen die nodig zijn om deze versterkte spanningvariaties om te zetten in corresponderende temperatuurvariaties gebeuren allemaal intern in de sensor. Hierdoor kan het uitgangssignaal rechtstreeks gebruikt worden om in temperatuurvariabelen om te zetten

met een eenvoudige formule (zie verder). Het gebruik is dus heel eenvoudig, en er is geen kalibratie nodig.

In de sensor zit een microchip. Je mag deze sensor dan ook niet blootstellen aan te grote electrische velden (let op met statische electriciteit).

## De TMP36 sensor: gebruikslimieten

Om het even welke sensor je nodig hebt, het is altijd van groot belang om na te gaan of de omstandigheden van de metingen binnen de begrenzingen vallen die eigen zijn aan die sensor. Deze limieten zijn altijd gespecifieerd door de fabrikant in een online 'datasheet'. Enkele belangrijke 'stats' die meegegeven worden bij de TMP36, zijn hieronder weergegeven.

| Stat | Limieten |
|---|---|
| Afmetingen | 0.2" x 0.2" x 0.2" |
| Temperature range | -40°C tot 150°C |
| Foutmarge | 1°C (als je binnen de opgegeven temperatuur range blijft) |
| Prijs | € 1,50 - € 2,00 |
| Output range | 0.1V (-40°C) to 2.0V (150°C) but accuracy decreases after 125°C |
| Power supply | 2.7V to 5.5V only, 0.05 mA current draw |

Merk op dat de output range (spanning) alleen maar positieve waarden heeft, wat de verwerking iets gemakkelijker maakt.

**Tip:**
Om te werken met een bepaalde sensor, kijk altijd naar de **datasheet** van de fabrikant, en ga niet gewoon op internet zoeken naar allerlei info over deze sensor. Iedereen kan om het even wat op internet zetten, en er is geen controle op juistheid.

## De TMP36 sensor: hoe gebruiken?

- Verbind de linkse pin met een spanningsbron (2,7-5,5V).
- Verbind de rechtse pin met de grond (aarding, nul).
- Verbind de middenste pin met de analoge Arduino pin die de meetwaarde moet ontvangen.
- Voer de meting uit, je krijgt een reeks output waarden $V_{out}$ in mV (milliVolt).
- Pas onderstaande formule toe op de output waarden. De resultaten drukken de temperatuur uit in °C.

Temperatuur = ($V_{out}$ – 500) / 10

**Opmerking**
Wanneer meerdere sensoren aangesloten zijn op het systeem, gebeurt het soms dat de temperatuurmetingen onrealistische waarden opleveren. In dat geval is er interactie opgetreden tussen de TMP sensor en een andere. Je kan dit vermijden door in de code na elke meting de sensor te deactiveren, en een delay tussen beide metingen in te bouwen.

# COMMUNICATIE MET SENSOREN VIA I2C OF SPI

Alvorens de volgende sensor te bespreken, dienen we iets te weten over de communicatie die plaatsvindt tussen een microprocessor en de aangekoppelde onderdelen. Er bestaan meerdere types communicatie systemen (of communicatie interfaces). In deze vorming gebruiken we de volgende twee:

| | I2C bus | SPI bus |
|---|---|---|
| Hoofdkenmerken | 1 master 1 slave<br>2 verbindingen nodig | 1 master 1ofmeer slaves (CS)<br>4 verbindingen nodig |
| Voordelen<br>Nadelen | Eenvoudig (2 pins)<br>Traag bij veel data<br>Goedkoop<br>Hoger energie verbruik | Iets complexer (4 pins)<br>Sneller<br>Geen limiet woordgrootte<br>Laag energie verbruik |
| Gebruik in deze vorming | Temp sensor (TMP36)<br>Druk sensor (BMP280) | SD card |

De communicatie systemen verschillen in de manier waarop uitgewisselde databits erkend worden als start en stop bits en als data bits. Met andere woorden: er zijn regels binnen de I2C bus en SPI bus waardoor de processor ‘weet’ wanneer 0 en 1 stroompjes moet geïnterepreteerd worden als het begin en einde van uit te lezen data die van de sensoren komen.

Beide systemen zitten zo in elkaar dat een tijdsignaal (SCL = Serial CLock, tijdspulsen met vaste frequentie) als achtergrond dient om datasignalen uit te lezen.

Beide systemen maken het ook mogelijk dat een ‘master’ (de microcontroller) en één of meerdere ‘slaves’ (de sensoren of andere perifere apparaten) met elkaar communiceren. Zo kan de master als het ware bevel geven aan een slave om te beginnen met data leveren, alsook om ermee te stoppen.

Het is niet het opzet van deze cursus om de systemen in detail uit te leggen. Hieronder is een beperkte uitleg voorzien die nuttig is voor de oefeningen van de vorming zelf. Een minimaal inzicht is immers vereist om de sensoren correct aan te sluiten op de Arduino microporcessor.

## I2C bus

I²C = **Inter-Integrated Circuit** (uitspraak: i kwadraat c)

**Wat?**
De microprocessor gebruikt een bepaalde combinatie van SCL-signaal/data-signaal als de start van te registreren data. Vanaf dan begint het bezorgen en uitlezen van de sensordata. Een andere combinatie van SCL-signaal/data-signaal staat voor een ‘stop’. Dan stopt de microprocessor met het uitlezen van data van de sensor, en de sensor stopt met ze door te geven.

Het uitwisselen van start-stop-bevelen en van meetdata tussen de microcontroller en de sensor gebeurt over 1 lijn (SDA in beide richtingen). De data kunnen niet gelijktijdig van master naar slave EN omgekeerd gebeuren.

Er zijn slechts 2 pins in gebruik voor data uitwisseling, en beiden kunnen in beide richtingen bits doorgeven:

- Het klok signaal **SCL** (Serial CLock)
- Het data signaal **SDA** (Serial DAta)

Uiteraard zijn er daarnaast ook 2 pins nodig voor de stroomvoorziening van de sensor:

- **5V in** (spanningsbron)
- **GND** (aarding, nul)

**Master**
De micropocessor wordt de ‘master’ genoemd, want hier wordt controle uitgeoefend op de I2C bus. De processor geeft het start en stop signaal en zend het SCl signaal uit.

**Slave**
Elke sensor die we verbinden met de microprocessor wordt een ‘slave’ genoemd. De sensor luistert als het ware naar de bevelen van de processor (master), en geeft zijn data door aan die processor.

## SPI bus

SPI = **Serial Peripheral Interface**

**Wat?**
In dit communicatie systeem is er gelijktijdig informatie overdracht van de master naar de slave en omgekeerd. Er zijn 4 lijnen vereist voor data-uitwisseling.
De communicatie start met het definiëren van een kloksnelheid, via het signaal op de SCL lijn. Vervolgens wordt een andere lijn (CS) gebruikt door de microprocessor (master) om één van de aangesloten sensoren (slave) te activeren, en te beginnen met data uitlezen. Dit gaat door totdat de master (terug via de CS lijn) het signaal geeft om te stoppen met data uitlezen.

Er zijn 4 pins nodig voor data-uitwisseling:

- **SCL** of **SCK** : seriële clock
- **MOSI** : Master output slave input: op deze lijn wordt de data verzonden die de master uitzendt en die bij de slave ontvangen wordt. Niet in omgekeerde richting.
- **MISO**: Master input slave output: op deze lijn wordt de data verzonden die de slave uitzendt en die de master ontvangt. Niet in omgekeerde richting.
- **CS**: Chip select (of SS Slave Select): Deze lijn dient om één van de slaves te activeren. Wanneer de CS lijn laag is (nul Volt), dan start de activering. Wanneer de CS hoog is (5 Volt), dan wordt deze slave stilgelegd (stop). Elke slave moet een unieke pin hebben voor de CS lijn.

Merk op dat het geven van start-stop bevelen door de master en het uitzenden van meetdata door de slave gellijktijdig kan gebeuren.

Uiteraard zijn er daarnaast ook 2 pins nodig voor de stroomvoorziening van de sensor:

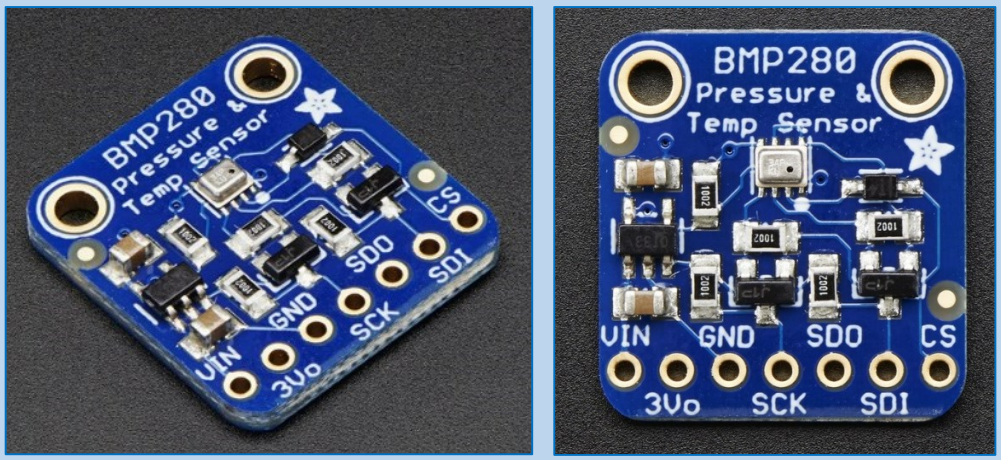
- **5V in** (spanningsbron)
- **GND** (aarding, nul)

# DRUKSENSOR

## BMP280: Inleiding

Hierboven lieten we je kennis maken met een eenvoudige sensor met 1 functie. In de elektronica is het echter gebruikelijk om het werk te beperken indien mogelijk, door samengestelde componenten te gebruiken.

De **BMP reeks** van Bosch zijn sensoren waarmee tegelijk de luchtdruk en temperatuur gemeten kan worden, en waar rechstreeks de hoogte kan uit afgeleid worden. De precisie van de luchtdrukmeting is ongeveer 1 hPa, en die van de temperatuur ongeveer 1°C. Dit maakt de sensor geschikt als barometer en om de hoogte te bepalen met ongeveer 1 meter foutmarge.




*De BMP280 sensor. Bron: Adafruit.*

## BMP280: Beschikbare pins

Er zijn 7 pins aan de onderzijde van de sensor. De eerste 3 zijn power pins:

- **Vin**
  Hier komt de stroom binnen. De sensor gebruikt 3 Volt DC. Je kan echter even goed 5V op deze pin zetten, want er is een ingebouwde spanningsregelaar. Je hoeft de standaard spanning van je microcontroller niet te wijzigen.
- **3Vo**
  Hier kan de stroom buitengaan, met een spanning van 3.3V. Je kan deze pin dus gebruiken als stroombron voor iets anders, met een maximum van 100mA. Zoniet, dan wordt deze pin niet aangesloten
- **GND**

De volgende 4 pins worden **'logic pins'** genoemd. Gebruik makend van de communicatie-interface I2C, hebben we er slechts 2 nodig. Bij communicatie via SPI, dat sneller data transfereert en minder stroom vraagt, hebben we 4 pins nodig.

Afhankelijk van de communicatie-interface (I2C of SPI, zie hoger) worden de pins van de BMP280 sensor anders gebruikt:

**Bij I2C communicatie:**

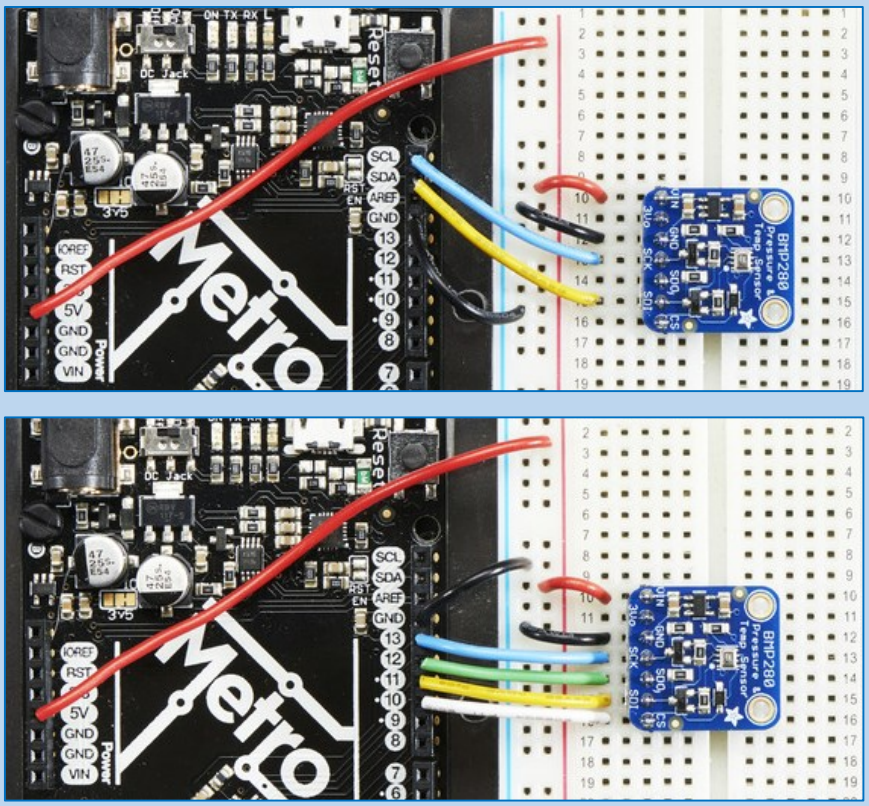
- **SCK** = Serial ClocK
  Deze pin stuurt tijdsgebonden pulsen naar de µcontroller. Deze moet bij onze Arduino verbonden worden met de SCL pin (de laatste pin langs de kant van de digitale, dit is de Arduino I2C clock pin). SCL staat voor Synchronize CLock. De tijdpulsen op dit signaal vormen de referentietijd voor het hele systeem (gebruikt als referentie voor data, maar ook voor het uitvoeren van de code zelf).
- **SDI** = Serial Data In/Out
  Deze pin stuurt data van de µcontroller naar de sensor EN omgekeerd. Deze moet bij onze Arduino verbonden worden met de SDA pin (de voorlaatste pin langs de kant van de digitale, dit is de Arduino I2C data pin). De data die uitgewisseld worden op dit kanaal zijn de meetgegevens zelf, maar ook de commando's van de µcontroller, en de signalen nodig om één bepaald 'slave' apparaat te activeren of deactiveren.

De andere twee pins blijven ongebruikt (niet verbonden).

**Bij SPI communicatie:**

- **SCK** = Serial ClocK
  Deze pin stuurt tijdsgebonden pulsen naar de µcontroller, idem als bij de I²C interface.
- **SDO** = Serial Data Out
  Deze stuurt data van de sensor naar de µcontroller (MISO = Master In Slave Out pin).
- **SDI** = Serial Data In
  Deze stuurt data van de µcontroller naar de sensor (MOSI = Master Out Slave In pin).
- **CS** = Chip Select
  Je hebt per apparaat dat je aansluit een unieke CS pin nodig. Ze worden gebruikt om te bepalen welke sensor op een bepaald moment geactiveerd moet worden.

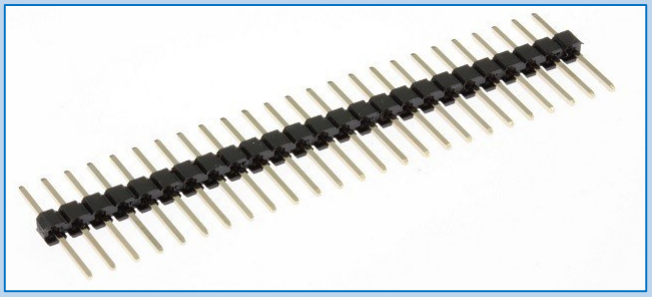
Alle 4 deze pins moeten elk verbonden worden met een unieke digitale pin (nrs 2-13) op de Arduino.
Wanneer je meerdere sensoren aansluit met SPI interface, dan kunnen alle SDO, SDI en SCK pins telkens gedeeld worden op 1 Arduino pin. De CS pin daarentegen moet voor elke sensor een unieke pin krijgen.




*De BMP280 sensor aangesloten op de arduino via I²C interface (boven) en via SPI interface (onder). Bron: Adafruit.*

## Werken met de BMP280

Op de figuur hierboven is de BMP sensor aangesloten aan de Arduino via een breadboard. Dit is een ideale manier om je satelliet in elkaar te knutselen en te testen, alvorens alles defintief aan elkaar te solderen. Om de sensor gemakkelijk te verbinden met de gaatjes van de breadboard, zit er een '**header**' bij: een reeks aan elkaar hangende pinnetjes met dezelfde afstand als de gaatjes van de sensor en het breadboard.



*Header. Bron: Adafruit.*

Wanneer de pins aangesloten zijn zoals hierboven, dan gebeurt alle verdere werk via de code. Je zal in de code de correcte functies van de verschillende aangesloten Arduino pins moeten definiëren, zodat die overeenkomen met de juiste BMP280 functies. Vervolgens kan

je de code schrijven om de gewenste meeresultaten te verkrijgen. Dit leren we verder in deze cursus.

### Als je zelf een druksensor kiest

In deze vorming werd de druksensor voor u uitgekozen. Als je echter voor een STEM project op school zelf op zoek gaat naar een druksensor, dan raden we aan om te letten op de volgende belangrijke eigenschappen:

- **Gevoeligheid (Sensitivity)**
  Wat is de minimale drukwijziging die kan gemeten worden door deze sensor?
- **Reactietijd (Response Time)**
  Hoe snel is de sensor?
- **Lineariteit (Linearity)**
  Hebben de output data van de sensor een lineair verband met de luchtdruk (voor de waarden range die je verwacht)?
- **Uiterste waarden (Range)**
  Wat zijn de minimale en maximale drukwaarden die door de sensor kunnen gemeten worden?
- **Hysteresis**
  Bij een dalende luchtdruk kan een sensor soms een lichtjes andere absolute meting geven dan bij een stijgende luchtdruk. Dit lichte verschil heet hysteresis, en is een maat voor een zekere traagheid waarmee een sensor de exacte waarde van de omgeving overneemt.

## µPROCESSOR OF µCONTROLLER

De µcontroller is het meest essentiële onderdeel van de Arduino UNO die in het vormingspakket zit. Deze component heeft een intern geheugen; het is hierop dat de code terecht komt die je zal schrijven voor je satelliet.

De Arduino UNO wordt verder besproken onder hoofdstuk 2b.

## BATTERIJ/POWER BANK

### Energie voor satellieten

Echte satellieten worden in de meeste gevallen voorzien van energie met zonnepanelen met fotovoltaïsche cellen die in serie geplaatst zijn. Fotovoltaïsche cellen zijn eigenlijk omgekeerd LEDs. Als er licht op invalt, wordt een kleine stroom gecreëerd. Gewoonlijk zullen de zonnepanelen eerst een batterij opladen, die vervolgens de nodige stroom voorziet voor de instrumenten van de payload. Dit is des te meer nodig wanneer een satelliet in een lage baan om de Aarde cirkelt, aangezien hij bij elke omwenteling een tijdje in de schaduw van de Aarde zit.

Het energieverbruik van de satelliet moet in detail berekend worden, zodat de stroomvoorziening er precies kan op afgesteld zijn.
In deze vorming zullen we enkel een batterij gebruiken, die vooraf ruim voldoende opgeladen wordt.

## Welke batterij gebruiken we?

Als stroombron voor onze payload gaan we kiezen voor een klassieke 9V batterij of een power bank.


*Een klassieke 9 V batterij en een standaard powerbank die via USB verbinding herladen kan worden.*

Beiden zijn lichte en goedkope batterijen, die overal verkrijgbaar zijn.

Volgende kenmerken kunnen van belang zijn voor een **9V batterij**:

- Niet elektronisch geregeld, dus het geeft altijd stroom wanneer de batterij aangelsoten is.
- Is iets zwaarder.
- Heeft een kleinere energiedichtheid (alkaline batterij). De payload kan dus minder lang voorzien worden van energie. Dit vormt geen probleem voor deze vorming, maar mogelijks wel in andere experimenten (bijvoorbeeld bij ASGARD Balloons for science).

Volgende kenmerken kunnen van belang zijn voor een **power bank**:

- Soms elektronisch geregeld. Het kan zijn dat de powerbank geen stroom geeft, wanneer de verbruiker die eraan hangt (onze payload) te weinig stroom vraagt. Dit is ingebouwd in sommige power banks om te voorkomen dat een klein lek de energie laat wegvloeien wanneer een aangesloten apparaat niet aanstaat. Bij aankoop moet men hierop letten.
- Is iets lichter.
- Heeft een grotere energiedichtheid (Lithium batterij).
- Kan steeds herladen worden via USB voeding.

# WEERSTANDEN

## Weerstanden?

Deze component is naar zijn functie genoemd. Een elektrische weerstand **beperkt de doorgang van elektrische stroom** en veroorzaakt ter plekke een gewenste vermindering van het geleidingsvermogen. Hoe hoger de weerstandswaarde hoe sterker de stroomdoorgang wordt beperkt. Geleiders zoals koper en aluminium hebben een heel kleine weerstand voor stroom. Isolators zoals pvc en glas geleiden vrijwel geen stroom en hebben dus een heel hoge weerstand. Weerstanden in elektronische componenten hebben een vooraf **bepaalde waarde** die daar ergens tussen in zit. De afkorting voor weerstand zoals gebruikt worden op stuklijsten en schema's is de: **R (van Resistance).**

Weerstand R wordt ook wel met het **Ω (ohm)** teken aangegeven, bijv. R = 512 Ω. Dit is eigenlijk de **eenheid** van weerstand.
De "gewone" weerstanden (met uitlopers of pootjes) zijn normaal gesproken van een kleurcode voorzien (gekleurde ringen) en zijn meestal niet erg ESD-gevoelig (ESD = electrostatic discharge). Daarom hoef je ze niet te bewaren in gesloten plastic zakjes, en kan je ze aanraken zonder dat er iets verandert.

## Gebruik

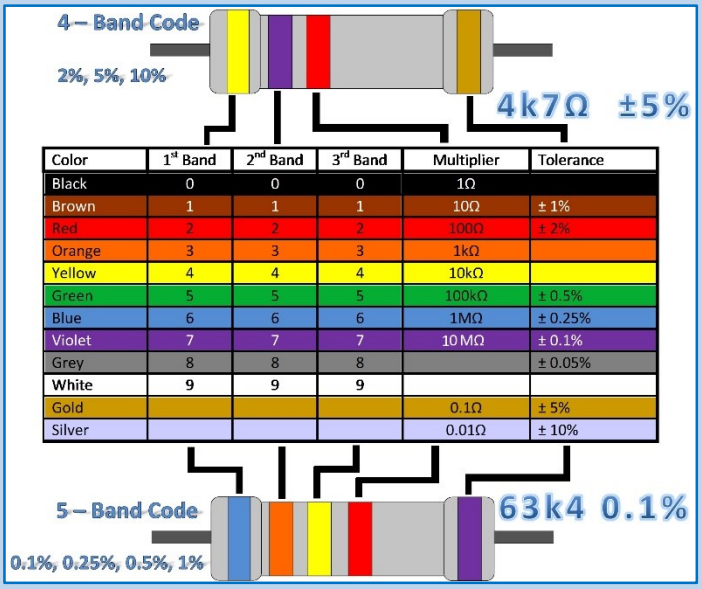
Weerstanden hebben we nodig om de stroom doorheen bepaalde (gevoelige) onderdelen of doorheen het hele systeem te begrenzen. Dit is bij een Arduino-gebaseerde satelliet van groot belang, want wanneer de maimale toegelaten stroom overshreden wordt, dan is onze µcontroller rijp voor de vuilbak! We leren verder in de cursus hoe we dit vermijden.

## Kleurcodes

### Weerstanden met vier ringen

- De eerste twee ringen bepalen het getal. Bijvoorbeeld: eerste band geel, tweede band paars: geeft basisgetal 47.
- De derde ring bepaalt de vermenigvuldigingsfactor, of hoeveel nullen komen achter het getal. Bijvoorbeeld: derde band rood betekent vermenigvuldigen met $10^2$ (of x100).
- De vierde ring is om de tolerantie of foutenmarge van de weerstand aan te geven. Bijvoorbeeld : vijfde band goud betekent dat de reële waarde van de weerstand (in Ohm) tot 5% kan afwijken tov de opgegeven waarde.
- De waarde van de weerstand in ons voorbeeld is dus 4700 (+ of – 235 Ω). Of anders geschreven: 4k7Ω + of – 5%.

De eerste ring zit het dichtst bij de draaduitloper en de laatste ring is breder gemaakt.




| Color | 1st Band | 2nd Band | 3rd Band | Multiplier | Tolerance |
|---|---|---|---|---|---|
| Black | 0 | 0 | 0 | 1Ω | |
| Brown | 1 | 1 | 1 | 10Ω | ± 1% |
| Red | 2 | 2 | 2 | 100Ω | ± 2% |
| Orange | 3 | 3 | 3 | 1kΩ | |
| Yellow | 4 | 4 | 4 | 10kΩ | |
| Green | 5 | 5 | 5 | 100kΩ | ± 0.5% |
| Blue | 6 | 6 | 6 | 1MΩ | ± 0.25% |
| Violet | 7 | 7 | 7 | 10 MΩ | ± 0.1% |
| Grey | 8 | 8 | 8 | | ± 0.05% |
| White | 9 | 9 | 9 | | |
| Gold | | | | 0.1Ω | ± 5% |
| Silver | | | | 0.01Ω | ± 10% |

*Betekenis van de kleurcodes bij weerstanden met 4 en met 5 ringen.*

Bij **weerstanden met 5 ringen** zijn het niet de eerste twee banden, maar wel de eerste drie banden die het getal bepalen. De vierde band is dan de vermenigvuldiger, en de vijfde band de tolerantie.

## KABELS

Het vormingspakket bevat 2 soorten kabels die essentieel zijn voor elk Arduino project:

### USB verbinding voor Arduino UNO

Het type kabel: USB 2.0 A naar B (male/male). Dit is het type kabel waarmee Arduino verbonden wordt met de laptop of desktop computer. De meeste printers gebruiken dezelfde kabel.

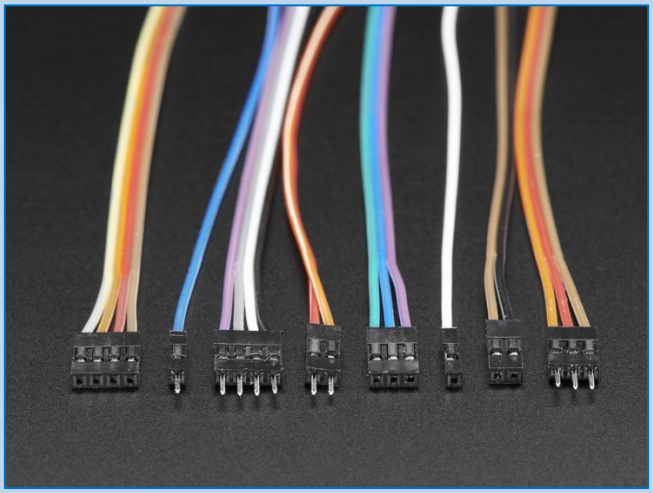
Wanneer deze kabel aangesloten wordt op de computer en op de Arduino, dan zal er een LED oplichten op de Arduino. Dit is het signaal dat de µcontroller van stroom voorzien is. Behalve stroom worden uiteraard ook alle data (code (sketches), input, output data) uitgewisseld via deze kabel.

*USB aansluitingskabel waarmee de computer en de Arduino verbonden worden met elkaar. Bron: Sparkfun.*

## Gekleurde kabelset of jumper kabels

De serie gekleurde fijne kabels (in dit geval male/male) worden gebruikt om allerlei pins te verbinden. Ze worden ook soms jumper kabels genoemd.

Wanneer je nadenkt over het elektronisch circuit die je in elkaar knutselt, zal je dat gewoonlijk eerst schematische tekenen. De vele kleurtjes van de jumper kabels laten toe dat je het circuit vervolgens correct nabouwt met echte verbindingen op de breadboard.



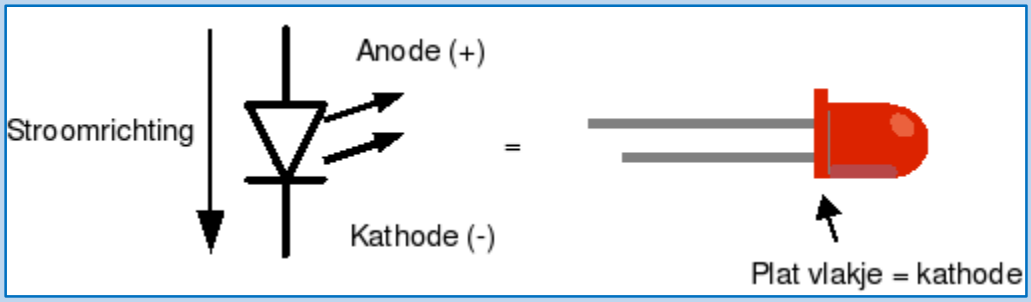
*Platte jumper kabels in met mannelijke (male) en vrouwelijke (female) uiteinden. Bron: Adafruit.*

# LEDs

## Wat is een LED?

LED is de afkorting van **Light Emitting Diode**, wat zoveel betekent als lichtgevende diode. En inderdaad vertoont een led ook een belangrijke eigenschap van een diode, namelijk dat de **stroom** slechts **in één richting** wordt doorgelaten; er kan alleen stroom lopen van de anode (positieve pool) naar de kathode (negatieve pool). Dit is een belangrijk verschil tussen een led en een gloeilamp, waarbij de polariteit niet uitmaakt.

Een ander verschil met de gloeilamp, is dat een diode de electrische stroom niet afremt (geen weerstand). Daarom moeten we bij het gebruik van een LED zelf de **stroom afremmen**, door er een **weerstand** voor of achter te plaatsen.




*Schemasymbool (links) en tekening (rechts) van een klassieke standaard LED.*

Bij de hier weergegeven “gewone” LED met twee aansluitdraden zijn de anode en de kathode als volgt te herkennen:

- De **anode** (de positieve aansluiting) is de langste draad.
- De **kathode** (de negatieve aansluiting) is de kortste draad. Verder is de kathode bij ronde leds vanaf 5 millimeter ook aangegeven door middel van een plat vlakje bij een van de draden.

## Een LED in het circuit opnemen

Houd er rekening mee dat er LEDs in talloze kleuren, vormen, maten en varianten verkrijgbaar zijn, zodat niet altijd direct duidelijk is wat de anode en de kathode is. Meerkleurige LEDs hebben bijvoorbeeld meer dan 2 draadjes.

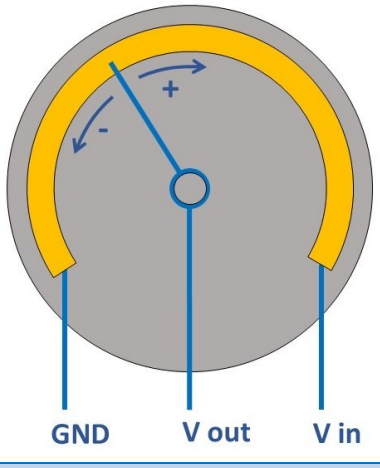
LEDs moeten in het circuit worden vergezeld van een weerstand (in serie geplaatst), om te vermijden dat er teveel stroom doorheen gaat. Het is aan de gebruiker om zelf de juiste weerstand te berekenen.

Er zijn dan ook twee waarden belangrijk bij het gebruik van een LED; Je vindt ze in de bijhorende datasheet:

- $U_{LED}$ is de **minimale spanning** die nodig is voor het verkrijgen van de halfgeleiderwerking. Bij een lagere spanning is er geen licht.
- De **maximale stroom** die door de LED mag gaan zonder dat hij kapot gaat.

## POTENTIOMETER

Een potentiometer of potmeter is een weerstand waarvan de weerstandswaarde kan aangepast worden door een draai- of schuifbeweging te maken. Ze worden gebruikt om een spanning gradueel te verkleinen naar keuze.




*Schema van een draaibare potentiometer. Bron: ESERO BE.*

## SCHAKELAAR

Een **tactiele schakelaar** (eentje die je aan/uit doet door er 1 keer op te duwen) van 6 mm groot en 4 pootjes van 2,5 millimeter wordt in deze vorming gebruikt om de stroom naar de Arduino aan en uit te zetten. We gebruiken hem samen met bepaalde weerstanden om onze Arduino te beschermen tegen blijvende schade door te hoge stroom (zie verder).

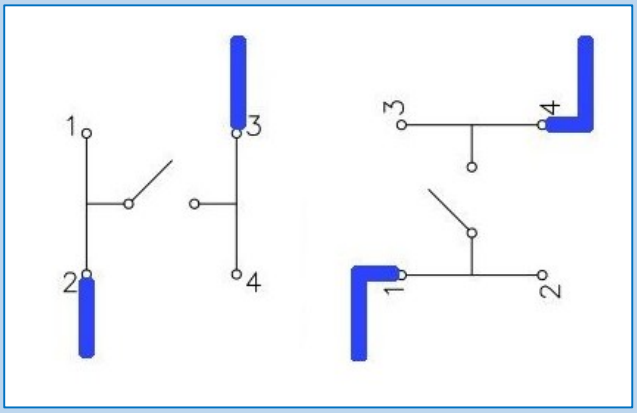



*Foto en schema van de tactiele schakelaar van de MC Hobby shop. Uit het schema blijkt dat je telkens de diagonaal tegenover elkaar liggende contacten moet gebruiken om de schakelaar in een circuit in te bouwen. Bron: Shop MC Hobby Belgium.*

De 4 pootjes passen precies in de pins van de Arduino, van andere componenten, en van de breadboard.

# 2b Arduino

## TERMINOLOGIE

Bij het gebruik van Arduino (en andere µcontrollers) hoort een uitgebreide terminologie of woordenschat. We zetten de belangrijkste termen hier op een rijtje:

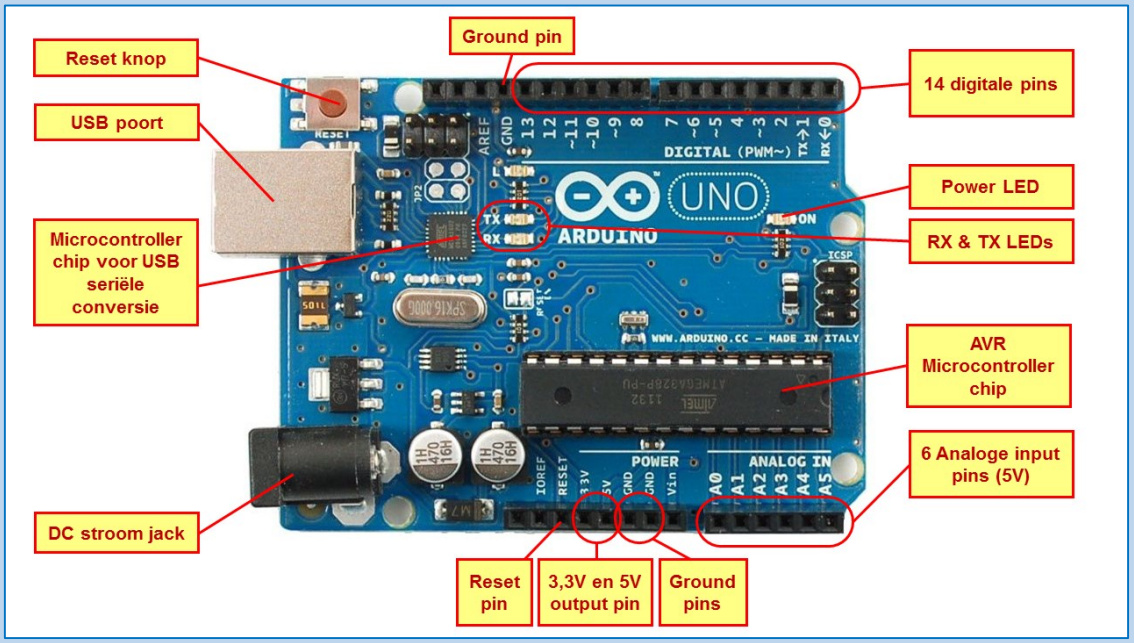
| Term | Uitleg |
|---|---|
| Master/Slave | Gewoonlijk wordt de µcontroller beschouwt als de 'Master': het centrale apparaat dat de commando's geeft aan de perifere apparaat. Deze laatste ontvangen de commando's, voeren ze uit, en geven data (het 'werk') terug. |
| Sketch | Dit is een volledig programma(tje) geschreven in het code-venster van de software, om vervolgens op te laden naar de Arduino controller. De code-taal is C.<br>Een sketch is eigenlijk de Arduino term voor een bestand (file). Het wordt opgeslagen met de extensie .ino (vroeger .pde). |
| Bibliotheek (Library) | Dit is een stukje code met een zekere functie die je online vindt. Je verwijst naar een bibliotheek in je sketch helemaal in het begin van de code, via het commando "`include`".<br>Bibliotheken worden gebruikt om te vermijden dat je voor bepaalde (courrant gebruikte) functies telkens weer 'het wiel moet uitvinden', of om het gebruik van bepaalde onderdelen gemakkelijker te maken.<br>Bibliotheken krijgen gewoonlijk een naam die verraad over welke functies het gaat.<br>De bibliotheek staat bewaard ergens op je laptop, je code geeft een verwijzing er naartoe. |
| Shield | Dit is een kunstof plaatje (Printed Circuit Board of PCB) met vooraf ingebouwde verbindingen en componenten/apparaten of door uzelf gemonteerde componenten/apparaten, die in zijn geheel aan de Arduino UNO gekoppeld wordt. Met dergelijke shields kan met in 'verdiepingen' werken om de Arduino satelliet uit te breiden met meerdere functies. |
| compiling | Dit is het proces waarbij de code zoals wij die ingeven (een 'human readable' versie van het programma) omgezet wordt in een code die door de chip begrepen wordt, en dus kan uitgevoerd worden. Dit proces is het werk van de software. |
| Syntax | Een taal heeft grammatica en leestekens om communicatie correct en duidelijk te laten verlopen. Dat is ook het geval bij de programmeertaal C++. Hier wordt dat syntax genoemd. Het is de set regels om correct in de programmeertaal te schrijven. |
| Hex code | De software van Arduino communiceert met jou via een C-code. Dit is de programmeertaal waarin je sketch en de bibliotheken in geschreven zijn. Dit wordt ook wel een "human readable" medium genoemd.<br>Maar uiteindelijk zal de software je programma omzetten in een totaal andere code alvorens die naar de Arduino chip gestuurd |

| | |
|---|---|
| | wordt. Deze finale code – niet leesbaar voor 'humans', wel voor microcontrollers – wordt de hex code genoemd. |

## HARDWARE ARDUINO UNO

Aangezien de Arduino producten '**open source**' zijn, kan om het even wie klonen en varianten maken van Arduino boards, en deze verkopen. Dat is perfect legaal.
Vele klonen zijn van even goede kwaliteit als het origineel.

Welke **onderdelen** vinden we terug op een Arduino UNO board? Hieronder worden de belangrijkste overlopen.




*Belangrjkste onderdelen van de Arduino UNO board. Bron: ESERO BE..*

### AVR Microcontroller chip of microprocessor

Hierop worden de codes bewaart die je maakt in de Arduino IDE software. Het is het 'brein' van de Arduino. Het zorgt voor de uitvoering van de code wanneer het programma gestart wordt.
De chip heeft een flash geheugen van ... MB.

### Microcontroller chip voor USB seriële conversie

Deze chip is vereist om de communicatie tussen de computer (met IDE softwae) en de Arduino mogelijk te maken.

### USB poort

Aansluiting voor de USB kabel die de Arduino UNO verbindt met de computer. Deze verbinding zorgt voor de stroomvoorziening en voor dataverkeer tussen de Arduino en de computer.
Een alternatieve stroomvoorziening is mogelijk via de DC stroom jack.

### DC stroom jack

Hierop kan je bijvoorbeeld een standaard 9V batterij op aansluiten (als je de vereiste aansluitkabel hebt) om de Arduino van stroom te voorzien wanneer deze niet meer aan de computer hangt. Dit is het geval tijdens de vlucht van je satelliet. De batterij moet een spanning leveren tussen 7V en 12V.
Als alternatief kan je de via de USB kabel een opgeladen power bank aansluiten.

### 6 analoge input pins

Hier komen analoge data binnen van aangesloten sensoren (max 5V).

### 14 digitale pins

Deze pins kunnen gegevens ontvangen van digitale aangesloten apparaten, en ook gegevens versturen naar aangesloten apparaten. In beide gevallen gaat het over spanningsvariaties tot maximaal 5V (bits: spanning nul of max).

**Digitale pins met tussenwaarden**
De digitale pins waar **~** bij staat (de pins **3, 5, 6, 9, 10, 11**) kunnen gebruikt worden voor lagere spanningswaarden dan 5V. Een pin waar geen ~ bij staat zal op bepaalde momenten 0V of 5V geven, geen andere waarden.

Maar hoe geeft een pin met ~ een ander voltage? Arduino kan op digitale pinnen immers alleen maar 5 of 0 volt geven? Eigenlijk wisselt hij duizenden keren per seconde tussen 0 en 5 volt. Door de intervallen van 0V (of de intervallen van 5V) korter of langer te maken, zal het gemiddeld uitgangssignaal een waarde zijn tussen 0 en 5V, bijvoorbeeld 2.5V wanneer biede intervallen exact gelijk zijn.

De tijd is hierbij een belangrijke referentie. Het ‘kristal’ (metaalkleurig ovaal onderdeel) op het Arduino board zorgt ervoor dat de Arduino op een vaste kloksnelheid draait.

**Pin 0 en 1: TX en RX**
T staat voor Transmit, R staat voor Receive. Ze kunnen gebruikt worden voor seriële communicatie tussen de Arduino en een ander apparaat. Wanneer data via zulke communicatie binnen of buiten gaan, dan gaan de overeenkomstige LEDs (TX en RX) flikkeren. Dit is een belangrijke aanwijzing bij foutopsporing om te controleren of je Arduino daadwerkelijk informatie aan het verzenden/ontvangen is.

Standaard worden de digitale pins als input pins beschouwd. Indien je ze als output wilt gebruiken, dan moet dit zo bepaald worden in de code.

### 3,3V en 5V output pins

Deze pins dienen als stroombron voor aan te sluiten apparaten, respectievelijk met een spanning van 3,3V en van 5V.

### Reset knop en Reset pin

Als je op deze knop duwt, dan zal de Arduino het opgeslagen programma terug van het begin uitvoeren. De code op de microprocessor wordt NIET gewist.
Precies hetzelfde gebeurt wanneer er 0V op de reset pin gezet wordt.

### Ground pins

De GND pins geven toegang tot de laagste spanning van het hele systeem.

### Sinken en sourcen

Wanneer in een circuit de Arduino zelf een bron van stroom is voor een perfieer onderdeel, dan spreekt men over '**sourcen**'. De totale stroom die je kan sourcen met 1 Arduino UNO is **200 mA**. (max 40 mA per pinnetje).

Wanneer de stroombron een extern apparaat is, dan dient de Arduino als eindpunt van de stroom via de GND pinnetjes. Dit is '**sinken**' Er zijn echter 2 GND pinnetjes die hiervoor kunnen dienen. Bij sinken kunnen we dan ook 2 x 200mA door de Arduino sturen, dus een totaal van **400 mA**. (max 40 mA per pinnetje).

### Power LED

Deze LED verraadt of er al dan niet spanning staat op de Arduino, dus of hij 'aan of uit' is.

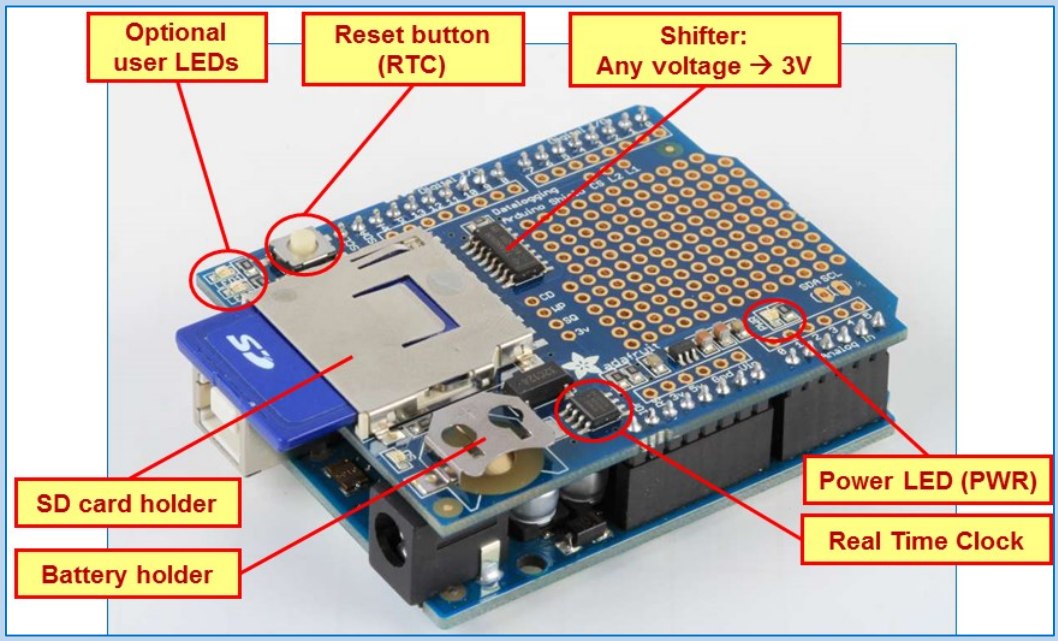
## SHIELD AANSLUITINGEN – SD CARD DATALOG SHIELD

### Wat is een shield?

Merk op dat ter hoogte van de digitale en analoge pins telkens een zwart kunstof strookje zit met gaatjes. Alle Arduino board (ook de kopieën van andere makers) hebben diezelfde layout. Daardoor passen allerlei shields vanzelf op het board. Een shield is een soort 'extra etage'. Dit wordt gebruikt om extra onderdelen aan te koppelen, zoals bijvoorbeeld: touchscreen, SD kaartlezer, SIM kaartlezer, motoren, enz.

### Datalog shield

In deze vorming beschik je over een datalog shield waarin een SD kaartje past. Dit hebben we nodig om meetdata op te slaan. De Arduino UNO chip heeft een klein intern geheugen van enkele honderden bytes. Dit is ruim onvoldoende om de meetgegevens allemaal te bewaren tijdens de vlucht. Daarom is een SD kaart noodzakelijk.




*Shield voor SD kaart met real time clock (RTC), hier met SD kaart erin en reeds gesoldeerd op een Arduino UNO board. Bron: ESERO BE.*

- **RTC:**
  De SD kaart shield uit deze vorming heeft dus een eigen Real Time Clock, die de data op je SD kaart kan voorzien van een tijdsreferentie.
- **Battery holder:**
  Je kan er een batterijtje in stoppen, zodat de klok autonoom jarenlang blijft lopen zonder aangesloten te zijn op een Arduino onder spanning.
- **SD card holder:**
  Elke SD/MMC geheugenkaart kan hierin. Wanneer je een micro SD kaart gebruikt, moet er een plastic adapter bij om te passen in het slot.
- **Voltage shifter:**
  Een SD kaart heeft een voedingsspanning van 3 Volt nodig. Hogere spanningen kunnen de kaart of de data beschadigen. Deze spanning shifter verzekert een 3V spanning op de SD kaart, ongeacht de binnenkomende spanning uit de Arduino.
- **Optional user LEDs:**
  Je kan uitgangen van de Arduino verbinden met L1 en/of L2 zodat je via de LEDs op de datalog shield kan zien wanneer het signaal op de betreffende uitgangen HIGH of LOW zijn.
- **Power LED (PWR):**
  Met deze LED kan je zien of de datalog shield van voedinsspannign voorzien is of niet.

Meer uitleg over het gebruik van deze datalog shield vind je verderop onder de titel “Data opslaan op SD kaart”.

# STROOM CONTROLEREN

## De stroomsterkte begrenzen

De datasheet van de Arduino vermeldt enkele belangrijke **beperkingen** in verband met **stroomsterkte** in de bijhorende datasheet:

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

Deze maximale stroomsterktes moeten gerespecteerd worden, anders zal de microcontroller beschadigd zijn.
- Er is 1 VCC pin. Deze Arduino kan in totaal dus maximum 200 mA doorsturen.
- Er zijn 2 GND pinnen. Deze Arduino kan dus in totaal maximum 400 mA ontvangen.

Om zeker te zijn dat de maximale waarden niet overschreden worden, moet er gewerkt worden met weerstanden in het circuit, in serie met de verbruiker (LED, Sensor, ...). Een dergelijk weerstand heet de **voorschakelweerstand**.

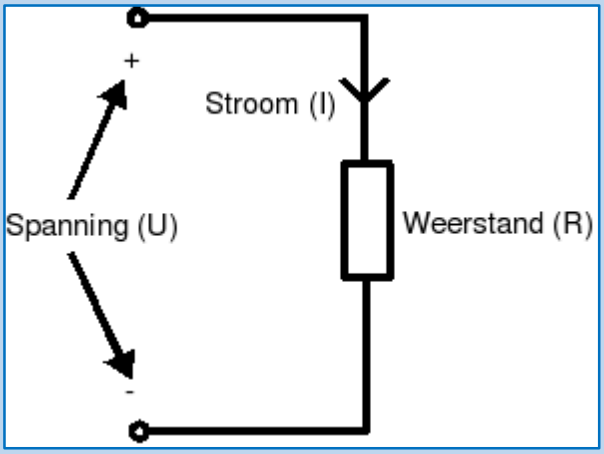
## Hoe bereken je de voorschakelweerstand?

Uitgangspunt bij de berekeningen is de **wet van Ohm**:

$$U = I \times R$$

U is de spanning in volt,
I is de stroom in ampère,
R is de weerstand in ohm.

Deze formule vertelt je de spanning over een bekende weerstand als er een bekende stroom doorheen wordt gestuurd.

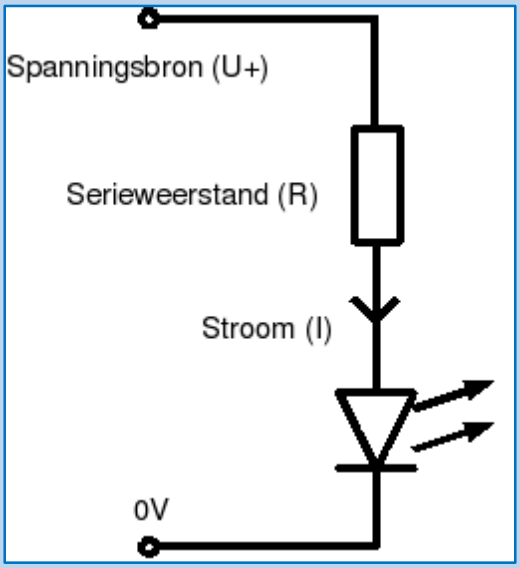



*Schema van een circuit dat de wet van Ohm illustreert. Bron: FabLab Klein-Brabant.*

Maar wat te doen als je niet de spanning wilt weten, maar de weerstand, of de stroom? Geen probleem, met een beetje brugklaswiskunde kan de wet van Ohm worden omgerekend tot de volgende formules:

**I = U / R** en **R = U / I**

Het vorige schema geeft de situatie aan voor de wet van Ohm, maar er ontbreekt nog een stroomverbruiker: **de LED**. Deze laatste wordt in serie geschakeld met de weerstand.




*Schema van een circuit met een weerstand en een LED in serie. Bron: FabLab Klein-Brabant.*

Hierbij is de **volgorde** van led en weerstand niet van belang, zolang de led maar in de juiste polariteit is aangesloten. Dat laatste betekent dat de +pool van de led moet aangesloten worden op de +pool van de spanningsbron, en idem met de -pool.

Verder moet de **spanningsbron** een **hogere spanning** leveren dan de nominale spanningsval van de LED, anders kan er al helemaal geen stroom lopen. Met “nominale spanning” bedoelen we de effectieve spanning die continu op de LED staat om de LED te laten functioneren.

In dit **voorbeeld** gaan we uit van een rode LED met een nominale spanningsval van 1,9 volt. Als we dus een voedingsspanning U+ van 5 volt nemen, zou het goed moeten gaan, want 5V is groter dan 1,9V. Als gewenste stroom I kiezen we 20 mA. Nu moeten we alleen nog de weerstand berekenen.

Nu zou je misschien denken dat je gewoon de waarden voor spanning (5V) en stroom (0,02A) in de wet van Ohm kunt invullen, maar dat is niet helemaal juist. Je moet nog rekening houden met de **nominale spanningsval over de LED**, die we $U_{led}$ noemen. Deze laatste spanningsval moet van de voedingsspanning U+ worden afgetrokken om de spanning over de weerstand R te krijgen. In formulevorm ziet dit er zo uit:

$$R = (U+ - U_{led}) / I$$

Nu kun je alle getallen invullen:

$$R = (5 – 1,9) / 0,02 = 3,1 / 0,02 = 155 \text{ ohm}$$

Je neemt dus gemakshalve een weerstand van **180 ohm** (een algemeen verkrijgbare waarde) als **voorschakelweerstand**. Deze weerstand is hoger dan de berekende 155 ohm, dus de stroom zal naar verwachting wat lager zijn. Maar dit is geen probleem, want de lichtopbrengst van een LED hoeft vrijwel nooit een erg nauwkeurige waarde te hebben.

We willen zoveel mogelijk controle over stroomstertes. Dus, voor de aardigheid kunnen we nog even **uitrekenen** wat die **stroom** dan eigenlijk is bij 180 ohm in plaats van 155 ohm. Ook bij deze berekening moeten we weer de nominale spanningsval van de LED aftrekken van de voedingsspanning:

$$I = (U+ - U_{led}) / R = (5 – 1,9) / 180 = 17,22 \text{ mA}$$

Je ziet, de stroom is wat lager dan de gewenste 20 milli-ampère, maar dat is niet erg. Het verschil in lichtopbrengst zal nauwelijks zichtbaar zijn.

## LET OP!

Goed, je kunt nu de benodigde voorschakelweerstand voor een LED berekenen, maar er zit een addertje onder het gras! Als de spanningsval over de weerstand **$(U+ - U_{led})$ erg klein** wordt ten opzichte van de voedingsspanning, kan een kleine afwijking in de voedingsspanning of de nominale spanningsval over de LED een grote afwijking in de stroom veroorzaken.

Stel, je hebt een blauwe LED (met $U_{led}$ gelijk aan 3,6 volt), en een voedingsspanning van 3,7 volt. Als je een stroom van 20mA wilt hebben, bereken je de voorschakelweerstand dus als volgt:

$$R = (3,7 - 3,6) / 0,02 = 5 \text{ ohm}$$

Je neemt dus een weerstand van 5,6 ohm, waarbij de stroom (3,7 - 3,6) / 5,6 = 18 mA wordt. Prima ... of toch niet?

Laten we eens kijken wat er gebeurt er als de spanning **een paar tiende volt afwijkt**. Bij een spanning van 3,5 volt (dus 0,2 volt onder de verwachte voedingsspanning) zit je al onder de nominale spanningsval van de LED. Het ding zal dan **amper oplichten**. Nog erger wordt het als de spanning 0,2 volt hoger wordt; de stroom wordt dan (3,9 - 3,6) / 5,6 = 54 mA.

Oeps, dit is meer dan de meeste LEDS verdragen EN **boven de maximale stroom** dat de Arduino op een IO pin kan leveren! Waarschijnlijk zie je de LED dan korte tijd bijzonder fel oplichten, om even later voor altijd te doven ...

## Weerstanden combineren

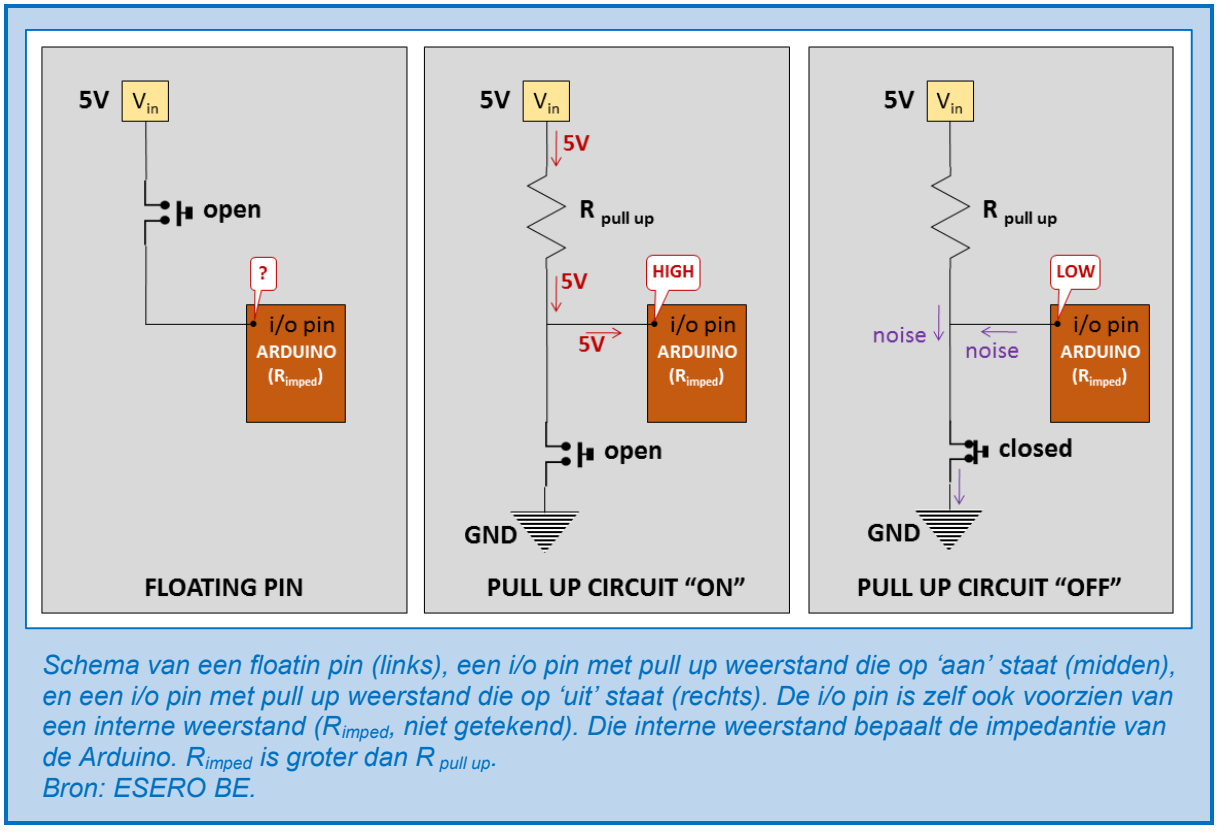
Als een weerstand met de gewenste waarde niet voorhanden is, dan kan je weerstanden in serie of parallel schakelen. De totale weerstand bereken je dan als volgt:

- Weerstanden in serie: $R_{totaal} = R_1 + R_2 + R_3...$
- Weerstanden in parallel: $1/R_{totaal} = 1/R_1 + 1/R_2 + 1/R_3 + ...$

## Pull up weerstand

In normale omstandigheden zijn we omgeven door electrische velden en statische electriciteit. Deze komen van talrijke apparaten rondom ons (mobiele telefoons bijvoorbeeld). Deze kleine stroompjes vinden hun weg in de i/o pins van de Arduino microcontroller. Het zijn allemaal **stoorstroompjes**, die ervoor zorgen dat er dikwijls signalen zijn in de Arduino pins wanneer de batterij of andere stroombron uitgeschakeld is (noise of ruis). Wanneer onze hoofdschakelaar uit staat, zijn er dus toch i/o pins die een signaal doorgeven aan de microchip. Deze pins noemen we floating pins.

Om deze bron van ruis te vermijden, gaan we tussen de batterij en de i/o pin een **zijsprong naar GND** voorzien waar ongewenste stroom kan wegvloeien. Uiteraard moet dan wel een weerstand voorzien worden tussen de batterij en de GND, om te verhinderen dat de batterij leegloopt. Het concept wordt hieronder uitgelegd.




*Schema van een floatin pin (links), een i/o pin met pull up weerstand die op 'aan' staat (midden), en een i/o pin met pull up weerstand die op 'uit' staat (rechts). De i/o pin is zelf ook voorzien van een interne weerstand ($R_{imped}$, niet getekend). Die interne weerstand bepaalt de impedantie van de Arduino. $R_{imped}$ is groter dan $R_{pull\ up}$.*
*Bron: ESERO BE.*

- Wanneer de schakelaar simpelweg tussen de batterij (5V) en de i/o pin zit, dan veroorzaken stoorstroompjes een onzekere toestand op de i/o pin. De input spanningwaarde op deze pin varieert daardoor ('floating'), en de digitale waarde kan daarom afwisselend 'HIGH' of 'LOW' zijn. Vandaar het vraagteken bij de **floating pin** hierboven.

We verbinden nu de batterij met de GND en de i/o pin. Op deze verbinding zetten we een schakelaar en een weerstand.

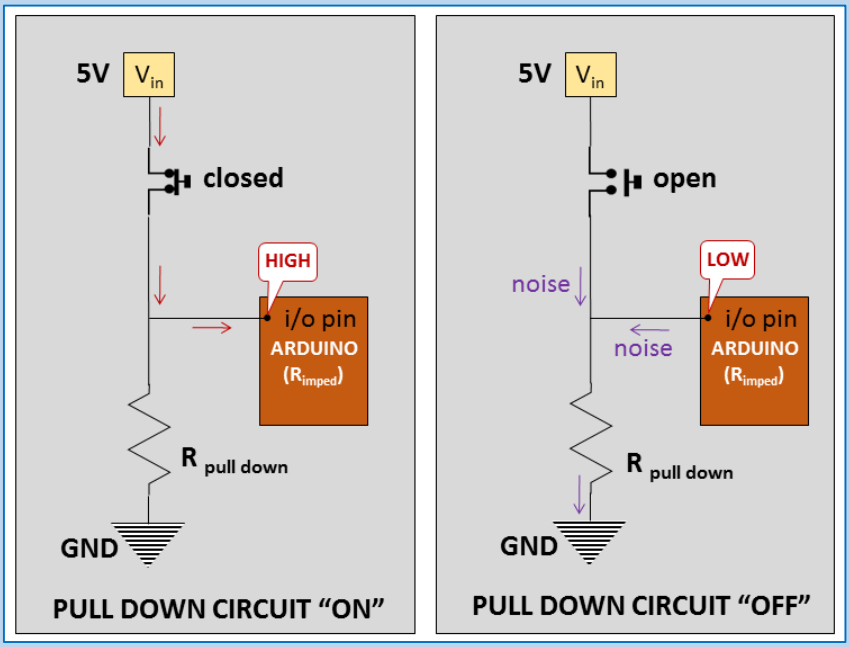
- Als we deze schakelaar open zetten, dan kan de stroom enkel naar de i/o pin lopen. Dan is de waarde van de **i/o pin "HIGH"** of 5 Volt.
- Als de schakelaar dicht staat, dan loopt de stroom van de batterij naar de aarde GND. De weerstand $R_{pull\ up}$ zorgt ervoor dat de batterij niet leegloopt. Een interne weerstand in de microchip (hoogohmige $R_{imped}$) zorgt ervoor dat de stroom wel degelijk naar GND loopt, en niet naar de Arduino. De logische waarde van de **i/o pin** is dan **"LOW"** of 0 Volt.

In deze opstelling moeten we dus de schakelaar uit zetten om de Arduino van stroom te voorzien. De weerstand die we hier gebruiken is de 'pull up' weerstand. Hij zorgt ervoor dat de i/o pin de hoogste waarde heeft (HIGH, 5V) in plaats van een 'floating' waarde. De waarde wordt als het ware opgetrokken (pull up).

Tegenwoordig zijn **pull up weerstanden ingebouwd** in de Arduino pins. Je hoeft ze dus zelf niet te solderen in het circuit. Met de code in de software kan je de ingebouwde pull up weerstanden inschakelen.

## Pull down weerstand

Je kan het bovenstaande probleem ook anders oplossen. We zetten nu een weerstand tussen de i/o pin en de GND.




*Schema van een i/o pin met pull down weerstand die op 'aan' staat (links), en een i/o pin met pull down weerstand die op 'uit' staat (rechts). De i/o pin is zelf ook voorzien van een interne weerstand ($R_{imped}$, niet getekend). Die interne weerstand bepaalt de impedantie van de Arduino. $R_{imped}$ is groter dan $R_{pull\ down}$.*
*Bron: ESERO BE.*

Als de schakelaar open staat bij deze opstelling, dan zal er op de i/o pin zeker 0 Volt staan. De stoorstroompjes worden immers via de R pull down naar de GND geleid. Deze weerstand zal de de i/o pin dus **omlaag brengen (pull down)** naar de logische staat **"LOW".**

Zowel een pull up als een pull down weerstand hebben als functie Arduino pins op duidelijke waarden LOW (0V) of HIGH (5V) te houden, en 'floating' te vermijden.

## SOFTWARE

### Download

De software die je moet installeren op je computer om met de Arduino te werken heet "Arduino IDE". IDE staat voor Integrated Development Environment. Als je deze naam ingeeft in je favoriete zoekmachine, dan zal je meteen de downloadpagina van Arduino vinden. Daar kan je de software gratis downloaden (via de knop "just download").

Dit is de download link: www.arduino.cc/en/Main/Software

Kies je systeem (Windows, iOS, LINUX), en download het programma. Vervolgens kan je het ontvangen bestand unzippen, en via een dubbelklik op het installatiebestand (.exe) installeren op je computer.

### De interface


*Interface van de Arduino IDE met aanduiding van de onderdelen. Bron: ESERO BE.*

### Menubar

Onder de openklapmenu's van de menubalk kan je elke toepassing terugvinden die de software aanbiedt. De meest gebruikte zijn echter gemakkelijker te gebruiken via een knop op de toolbar.

De opties onder de verschillende menu's spreken meestal voor zich. Ze worden enkel besproken in deze cursus in de mate dat we ze nodig hebben voor bepaalde oefeningen (zie verder).

Wel is het even handig om bij het begin onder het menu "hulpmiddelen" ("tools") het juiste Arduino board en de juiste USB poort te kiezen, zodat de computer en de Arduino correct met elkaar kunnen communiceren:

- Tools > serial port > klik de juiste USB poort aan (waar de Arduino verbonden is met de computer)
- Tools > board > klik het Arduino board aan dat je gebruikt (in ons geval Arduino UNO)

## Toolbar


**Verify**
Deze knop klik je aan om de code die je geschreven hebt te **controleren op fouten**. We bedoelen hiermee: fouten t.o.v. de regels in de codetaal, waardoor deze sketch niets zal kunnen uitvoeren. Eens gecontroleerd, zal het programma in je code aanduiden waar fouten gevonden werden. Het is dan aan jouw om uit te dokteren wat er precies fout is, en wat het juiste alternatief moet zijn.


**Upload Using Programmer**
Hier kan je een afgewerkte sketch **opladen naar de Arduino**. Dit doe je best alleen maar na volgende handelingen:

a) Duid aan welke USB poort er moet gebruikt worden voor het opladen (op welke USB poort is je Arduino aangesloten) via de 'tools' menu. Dit doe je 1 keer bij het begin van je project.
b) Controleer je sketch eerst op fouten via de 'verify' tool.
c) Bewaar je sketch eerst met de 'save' tool. Het kan voorkomen dat het systeem blokkeert bij het opladen, waardoor je werk verloren gaat.

Wanneer je een sketch upload, dan wordt de vorige overschreven. Zo kan je ook de Arduino terug leeg maken.


**New Editor Window**
Met deze knop krijg je een **nieuw tabblad** met een **blanco sketch**, klaar om te vullen met een code. De software zal je vragen naar de naam en de locatie om deze nieuwe sketch te bewaren. Het wordt aanbevolen om de standaard locatie te gebruiken. In de naam zijn geen spaties toegelaten.
De naam van je nieuwe sketch staat nu als titel boven het nieuwe tabblad.


**Open in Another Window**
Een nieuwe sketch kan met deze knop geopend worden in een nieuw venster.


**Save**
Bij een klik op deze knop wordt de **openstaande sketch bewaard** onder de naam en de plaats die je bij het begin bepaald hebt. Als je geen specifieke plaats gekozen hebt om de sketch op te slaan, dan heeft de software die standaard bij je sketch bibliotheek bewaard. Dit is handig om al je eigen werk gemakkelijk terug te vinden vanuit het Arduino IDE menu. Je sketch komt dan namelijk terecht in: file > sketchbook.
Wanneer de sketch bewaard is, dan verschijnt er ‘done saving’ onderaan.
Het is een goede gewoonte om na elke kleine wijziging in je programma op de save knop te klikken.


**Serial Monitor**
Als je klikt op Serial Monitor, dan krijg je de **data** te **zien** in het onderste zwarte venster (het serial board) die door je Arduino worden uitgestuurd. De Serial Communication Line is in dit geval de USB kabel.
Je kan de snelheid bepalen van het uitlezen van data via de ‘Baud’ knop boven het zwarte venster. Standaard staat deze snelheid op 9600. Dat betekent: er kunnen 9600 bits per seconde verstuurd worden.
Ook seriële communicatie in omgekeerde volgorde is mogelijk. Via het send venster kan je iets ingeven dat naar de Arduino verstuurd wordt.
Uiteraard is er alleen maar iets te zien in het serial board wanneer dit ook in de lopende sketch bepaald werd.

## Code venster

Hierin schrijf je de volledige code.
De software veranderd automatisch de tekstkleur wanneer bepaalde codes herkend worden als ‘geldige’ commando’s of functies.

Je kan ook waar je maar wilt commentaar erbij schrijven in gewone schrijftaal, meestal over wat de code precies doet. Er zijn namelijk symbolen die de software duidelijk maken dat je uitleg wilt bijschrijven die zelf geen code is. Deze symbolen vertellen de software: “wat ik nu ga schrijven is geen code, en moet dus niet naar de compiler gestuurd worden.” Met compiler bedoelen we: het vertaalprogramma die je code omzet in commando’s die de Arduino begrijpt na uploaden.

Als je **1 regel commentaar** wilt toevoegen, dan open je deze commentaar met 2 slashes (//), en je eindigt door op enter te duwen (nieuwe regel):

```
Code // hier schrijf ik mijn commentaar, namelijk uitleg over de
code hiervoor.
Nog meer code
```

Als je dit gebruikt, zal je zien dat de software je bijgevoegde commentaar (inclusief de slashes zelf) in lichtgrijs zet. Dit toont aan dat de software het herkend heeft als commentaar.

Als je **meerdere regels commentaar** wilt toevoegen, dan gebruik je slash + sterretje (/*) om de commentaar te openen, en sterretje + slash (*/) om te sluiten:

```
Code
/*
Hier kan ik meerdere regels eigen commentaar schrijven.
Hier kan ik meerdere regels eigen commentaar schrijven.
Hier kan ik meerdere regels eigen commentaar schrijven.
*/
Nog meer code
```

Voor alle andere weetjes die je nodig hebt om een code te schrijven, verwijzen we naar een de oefeningen verder in deze cursus (“eenvoudige codes”).

### Seriële output venster

Zoals beschreven onder toolbar: hier worden de data zichtbaar die de Arduino uitstuurt. Hier wordt echter ook door het programma gecommuniceerd over fouten wanneer je een foutencontrole doet (verify), of wanneer er andere foutmeldingen ivm het systeem zijn.

In de gekleurde lijn onder het seriële output venster staat links een nummer. Dit is de regel waarin uw cursor staat.

## PROGRAMMEREN : ALGEMENE REGELS

Als je programmeert, dan moet je de regels volgen van de gebruikte programmeertaal. Deze set van regels heet “**syntax**”. Wanneer je de regels niet correct toepast, dan krijg je van de software een foutmelding op de plaats waar het fout ging. Het controleren op fouten gebeurt door te klikken op de knop “verify”.

Er zijn veel syntaxregels, maar in dit gedeelte overlopen we er slechts enkele die voor elke sketch van toepassing zijn. Je leert deze natuurlijk vooral gebruiken door zelf eenvoudige programmeer-oefeningen te maken (zie verder). Voor alle andere regels en mogelijkheden verwijzen we naar de oefeningen verder in de cursus.

### Set up functie

Elke code wordt vooraf gegaan door de “set up”.
Hier worden bepaalde dingen vastgelegd alvorens aan de operationele code (de code voor de eigenlijke opdracht die je Arduino zal moeten uitvoeren) te beginnen. Zulke dingen kunnen bijvoorbeeld zijn:

- Bepalen welke Arduino pin als output of input zal dienen
- te gebruiken variabelen vastleggen
- identificatie van de sketch die gecommuniceerd wordt naar de seriële monitor
- ...

Wat er in Set Up staat wordt slechts **1 keer uitgevoerd** bij de start van het programma.

De volledige setup moet geschreven worden tussen linkse en rechtse **accolade** (waar hieronder de c-tekens staan).

```
Void setup() {
Ccccccccccccccccccc
Ccccccccccccccc
ccccccccccccccccc
}
```

## Loop functie

De loop bevat de eigenlijke opdracht die de Arduino lijn per lijn zal uitvoeren. De loop code wordt **continu opnieuw uitgevoerd** totdat de gebruiker hem laat stoppen.

De volledige loop moet geschreven worden tussen linkse en rechtse **accolade** (waar hieronder de c-tekens staan).

```
Void loop() {
Ccccccccccccccccccc
Ccccccccccccccc
ccccccccccccccccc
}
```

Het woord **void** gaat vooraf aan Setup en aan Loop. Het betekent letterlijk ‘de leegte’. Het staat er omdat zowel de setup als de loop op zichzelf staande eenheden zijn, onafhankelijke stukjes programma die niet gekoppeld zijn aan andere programma’s.

De **accolades** zijn handig. Wanneer je de cursor ergens in je code plaats, dan laat de software meteen zien waar de afsluitende accolade van dat stukje code staat, door er een klein kadertje rondom teplaatsen. Hetzelfde geldt voor gewone haakjes die een functie openen en afsluiten.

## De punt-komma

Achter elk statement van de je code moet er een punt-komma gezet worden. Je geeft hiermee het einde van het statement aan.

## Variabelen

Om te programmeren heb je variabelen nodig. Variabelen bevatten twee onveranderlijke eigenschappen:

- Een naam
- Een data-type

Beide eigenschappen worden door de propgramma-schrijver vastgelegd, en veranderen daarna niet meer. Wat er verandert, is de inhoud van een variabele. Het bepalen van de eigenschappen gebeurt als volgt:

```
Datatype naam ;
```

Wanneer je het datatype ingeeft, dan krijgt deze automatisch een oranje kleur (IDE herkent het datatype dat je ingeeft).

De naam

- Mag niet enkel een nummer zijn, en mag niet met een cijfer beginnen
- Mag geen spaties of speciale tekens bevatten

- Mag wel een underscore (_) of hekje (#)bevatten
- Mag geen termen bevatten die keywords vormen in de programmeertaal
- Is best een naam die de variabele omschrijft
- Een conventie is: wanneer je naam uit meerdere woorden bestaat, schrijf de woorden gewoon aan elkaar in kleine letters, waarbij de eerste letter van elk woord een hoofdletter is.

Voorbeeld:

```
int aantalSeconden;
aantalSeconden = 5;
```

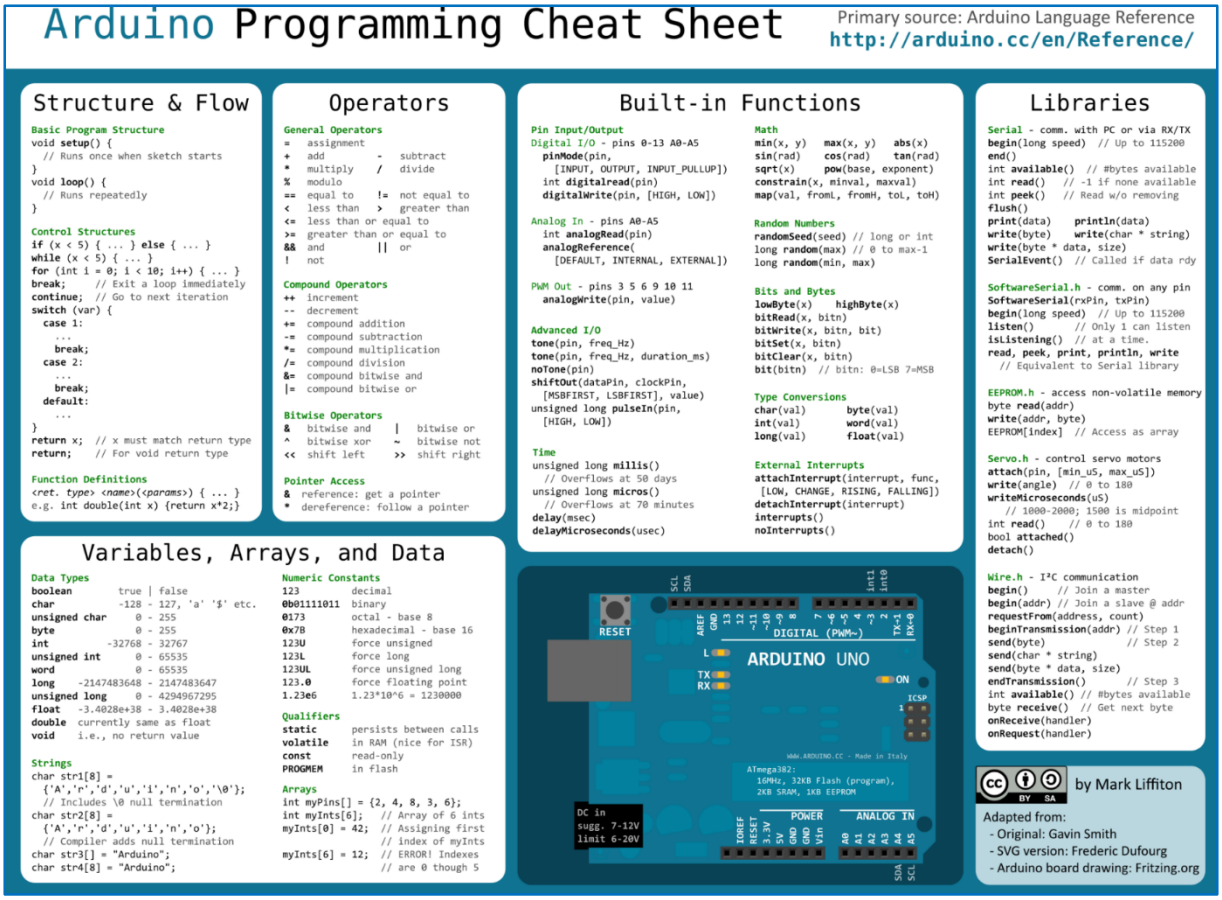
Deze variabele heeft de naam aantalSeconden, en is van het integer type. Integer variabelen zijn gehele getallen, zonder cijfers na de komma, en met een waarde tussen -32.768 en +32.768.

Let op: wat gebeurt er als de minimale/maximale waarden worden overschreden?
Als deze variabele op een zeker moment gelijk is aan +32.768, en er wordt +1 bij opgeteld, dan wordt verder geteld aan het andere uiterste van de toegelaten range. De nieuwe waarde is dan -32.768

In bovenstaand voorbeeld hebben we de variabele een initiële waarde gegeven, namelijk 5. Daarna kan ons programma beginnen lopen, en kan de variabele veranderen van waarde.

### Arduino Cheat Sheet

Arduino heeft een handig overzicht gemaakt van de meest gebruikte codes in A4-formaat. Dit heeft de “Arduino Cheat Sheet”. Het is erg handig om deze op te zoeken op internet, af te printen, en naast je te leggen tijdens het programmeren.




Let op: de programmeertaal is **hoofdlettergevoelig**. Als je bijvoorbeeld iets wilt sturen naar een digitale pin, dan gebruik je “digitalWrite”. Het commando digitalwrite doet niets. Het zal ook niet automatisch veranderen in oranje letters, omdat het niet herkend wordt.

### Sketch/bibliotheek downloaden

Zoals gezegd, de meeste Arduino gebruikers schrijven zelf geen sketches. Ze zijn allemaal te vinden op internet, meestal ook bij de fabrikant van het onderdeel dat je gebruikt, of bij grote leveranciers zoals Adafruit.

Als je een sketch donwloadt, dan kan je die best in het bibliotheken-mapje van de Arduino IDE zetten (sketch > import library > add library ...)

## EENVOUDIGE OEFENINGEN

Arduino is een open-source ontwikkelbord, waarmee het mogelijk (en relatief gemakkelijk) wordt om informatica te koppelen aan fysieke objecten. Het is dus een apparaat om fysische informatica makkelijk en toegankelijk te maken. De gebruiker moet echter leren om C codes te lezen en te schrijven.
Om de basis daarvan aan te leren, doen we hieronder enkele **eenvoudige oefeningen**.

# EENVOUDIGE OEFENINGEN : Een LED laten knipperen

## Opdracht

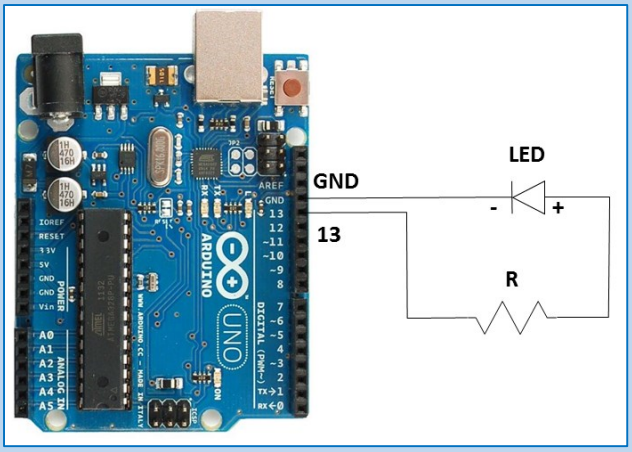
We gaan een LED laten aan en uit knipperen.
Je zou deze oefening kunnen doen met de ingebouwde LED van de Arduino UNO. Het heeft echter meer didactische waarde om met een extern aangesloten LED te werken, en het is cooler.

## Hardware

We hebben nodig:

- Een Arduino UNO
- Een LED
- Een weerstand van 220 Ω
- Breadboard
- Jumper kabeltjes
- Laptop + USB kabel

## Circuit




*Circuit voor de oefening "een LED laten knipperen". Bron: ESERO BE.*

We gebruiken een **digitale pin** van de Arduino (in dit geval 13), omdat de waarden van de LED binair zijn: aan of uit / 0V of 5V.
De **LED** mag om het even welke kleur zijn. De lange poot is de **anode** (+pool), en de korte is de **katode** (-pool).
De **weerstand** heeft geen vooraf bepaalde stroomrichting, en mag aangesloten worden op de + en – pool zoals je wenst.

## Sketch

In deze oefening gaan we de sketch niet zelf schrijven. Trouwens, de meeste Arduino gebruikers schrijven zelf geen sketches. Ze kopiëren bibliotheken of volledige sketches die te vinden zijn op het internet, en passen die aan volgens de noden.

Om een LED te laten knipperen (‘blink’), bevindt zich een standaard sketch in de voorbeeldbestanden van de Arduino IDE:

Ga via de menubalk naar:
File > Examples > basics > **blink**
Je krijgt onderstaande sketch:

```
/*
Blink
Turns on an LED on for one second, then off for one second,
repeatedly.

 This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

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

Laten we even enkele onderdelen van deze code bekijken.
Eerst voor de **Setup**:

```
int led = 13;
```

Hier wordt de variabele “led” gedefinieerd, met het datatype integer, en bij de start gelijkgesteld aan de waarde 13. We zullen deze waarde verderop gebruiken om 5V te sturen naar een bepaalde pin. De gewenste pin heeft immers een pin-nummer. En dat pin-nummer zal de waarde van onze variabele “led” zijn.

```
  pinMode(led, OUTPUT);
```

Hier bepalen we het volgende:
De pin met pin-nummer gelijk aan de waarde van “led” moet functioneren als output pin. Aangezien de waarde voodien gelijkgesteld werd aan 13, zal pin 13 dus gebruikt worden als output pin. Dat is logich, want we willen 5V naar buiten sturen op de pin waar onze led op zit.

De modus van de pin (pinMode) kan alleen maar op output of input gezet worden. Door het op **output** te zetten, zullen we verderop een waarde naar de digitale pin kunnen sturen met de functie '**digitalWrite**'.
Hadden we de pin modus op **input** gezet, dan zouden we dat kunnen gebruiken om een bepaalde waarde op deze pin te ontvangen en registreren met de functie '**digitalRead**'.

Nu gaan we verder naar de **Loop**:

```
  digitalWrite(led, HIGH);
```

Hier wordt de pin met pin-nummer gelijk aan de waarde van "led" (nog steeds 13) voorzien van maximale spanning of 5V (high). HIGH is een binaire waarde die door Arduino IDE herkend wordt, en daarom automatisch in blauw staat.
"digitalWrite" betekent: schrijf naar volgende pin (led) volgende waarde (HIGH).

```
  delay(1000);
```

Hier wordt gezegd: wacht nu 1000 milliseconden.

```
  digitalWrite(led, LOW);
```

Hier wordt de pin met pin-nummer gelijk aan de waarde van "led" (nog steeds 13) voorzien van minimale spanning of 0V (low).

## Upload

Klik nu op "upload" in de toolbar, en je zal zien dat de led begint te knipperen: 1 seconde aan, 1 seconde uit, enz. Dit wordt herhaald (de loop) zolang je het programma niet onderbreekt door de Arduino niet langer van stroom te voorzien.

## Verander het knipperpatroon.

Nu je de code in de blink sketch volledig begrijpt, kan je gaan spelen met het aantal milliseconden in de delay functie. Probeer maar.

## Maak de knippering zichtbaar in woorden

We willen niet alleen een led zien knipperen, we willen ook dat de Arduino zelf uitschrijft wat hij aan het doen is. Met andere woorden, telkens als de led aan of uit gaat, moet dit gecommuniceerd worden aan de computer. Dit kan erg nuttig zijn wanneer je wilt weten of iets aan of af gezet is dat je – in tegenstelling to onze led - niet kan zien.

```
int ledPin = 13; // een integer variabele krijgt de naam ledPin en
                    wordt gelijkgesteld aan 13

void setup() {
pinMode(ledPin, OUTPUT); // Pin 13 functioneert als output pin
Serial.begin(9600); // seriële monitor ontvangt gegevens met baud
                       rate 9600 (dwz: bits per seconde)
}
```

```
void loop() {
int a = 1000; // een integer variabele krijgt de naam a en wordt
                 gelijkgesteld aan 1000.
Serial.println("led aan"); // schrijf de boodschap "led aan" naar
                              de seriële monitor.
digitalWrite(ledPin, HIGH); // stuur 5V naar pin 13
delay(a); // wacht nu 1000 milliseconden
Serial.println("led uit"); // schrijf de boodschap "led aan" naar
                              de seriële monitor.
digitalWrite(ledPin, LOW); // stuur 0V naar pin 13
delay(a); // wacht nu 1000 milliseconden
}
```

### Extra oefening

Je kan een tweede led op pin 12 aansluiten (en de GND delen op dezelfde pin als de eerste led). Probeer nu led 1 en led 2 elk om beurt te laten aan en uit knipperen.

## EENVOUDIGE OEFENINGEN : Verkeerslicht

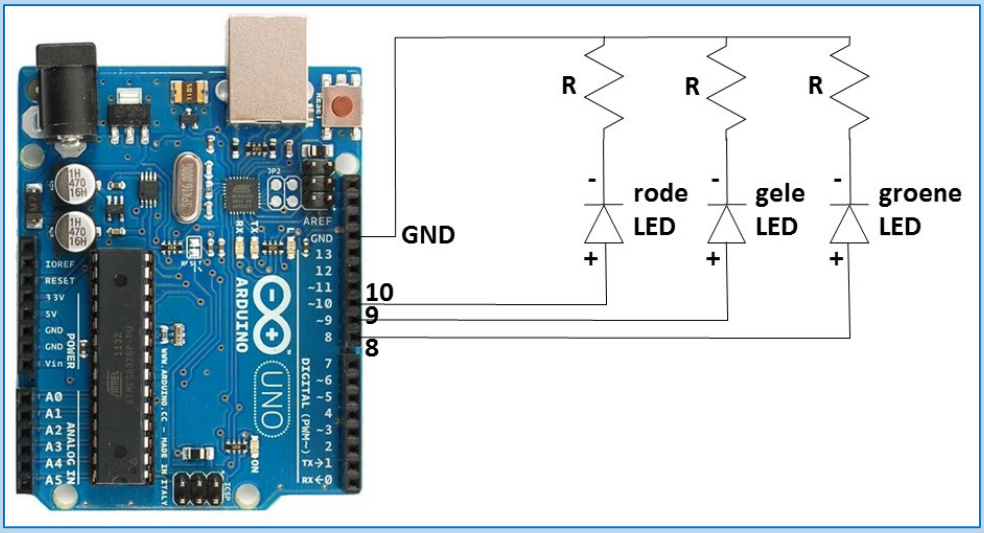
### Opdracht

We gaan een verkeerslicht in Engeland programmeren. Het verkeerslicht moet telkens groen en rood aangeven, en de overgang wordt aangekondigd met oranje. Zoals bij een echt verkeerslicht bljift deze cyclus zich eindeloos herhalen.

### Hardware

- Breadboard
- Rode LED, oranje LED, groene LED
- 3 weerstanden van 220 Ω
- Arduino UNO
- Jumper kabeltjes

## circuit




*Circuit voor de oefening "Verkeerslicht". Bron: ESERO BE.*

## Sketch

```
/*
Verkeerslicht:
Verkeerslicht gaat steeds op groen en op rood, met een overgang
die aangegeven wordt met een oranje licht. De duurtijd van rood en
groen licht kan geregeld worden met 1 parameter.
 */

int ledDelay = 5000; // deze delay gebruiken we tussenin de
                        kleurveranderingen
int redPin = 10;
int yellowPin = 9;
int greenPin = 8;

void setup() {
pinMode(redPin, OUTPUT);
pinMode(yellowPin, OUTPUT);
pinMode(greenPin, OUTPUT);
}
void loop() {
digitalWrite(redPin, HIGH);     // Zet de rode LED aan
delay(ledDelay);                // Wacht ledDelay milliseconden
digitalWrite(yellowPin, HIGH); // Zet de oranje LED aan
delay(2000);                    // Wacht 2 seconden
digitalWrite(greenPin, HIGH);   // Zet groene LED aan
digitalWrite(redPin, LOW);      // Zet rode LED uit
digitalWrite(yellowPin, LOW);   // zet oranje LED uit
delay(ledDelay);                // Wacht ledDelay milliseconden
digitalWrite(yellowPin, HIGH); // Zet oranje LED aan
digitalWrite(greenPin, LOW);    // Zet groen LED af
delay(2000);                    // Wacht 2 seconden
digitalWrite(yellowPin, LOW);   // Zet oranje LED af
```

```
}                                   // Nu herhaalt de loop zich
```

# TEMPERATUUR METEN

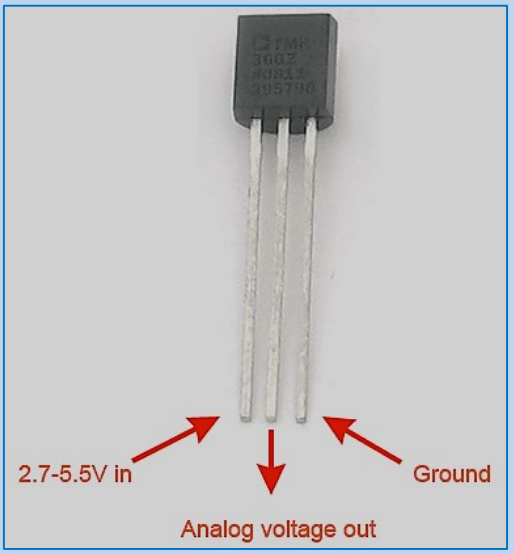
## Opdracht

Meet de temperatuur elke halve seconde met de meegeleverde TMP sensor. Laat de gemeten waarden verschijnen in °C in het seriële monitor venster. Warm de sensor op met je hand (en/of koel hem af met ijs) om te testen.

## Hardware

We hebben nodig:

- 1 Arduino Uno
- 1 breadboard
- 1 temperatuur sensor TMP36
- 1 weerstand 10kΩ (bruin/zwart/zwart/rood/bruin)
- 3 jumper kabeltjes




*De TMP36 sensor. Bron: Adafruit.*

## Van output voltage naar temperatuur

De output pin (analog voltage out) zal een waarde geven tussen 0 en 1,75 Volt, onafhankelijk van de input spanning (die 3,3V of 5V kan zijn).

De analoge pin van de Arduino (ADC) werkt met 10-bit waarden van 0 tot 1023. Deze worden eerst omgezet in milliVolt met deze formule:

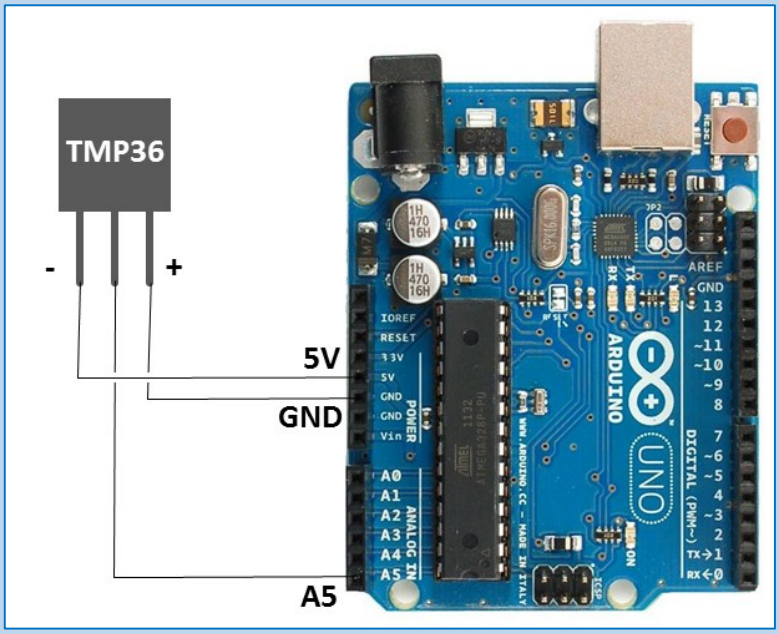
**Waarde in milliVolt** = (ADC-reading) x (input spanning/1024)

Input spanning kan 3300 mV of 5000 mV zijn.

Deze waarde in mV moet nu nog omgezet worden in temperatuur in graden Celcius. Hiervoor gebruiken we:

**Temperatuur in °C** = [(waarde in mV) – 500] / 10

## Circuit




*Circuit voor de oefening "De temperatuur meten". Bron: ESERO BE.*

## Sketch

In deze sketch komen we enkele nieuwe commando's tegen:

- **Float**
  Dit is een data type, waarin de variabele decimalen na de komma kan hebben. Voor een overzicht van data types: zie bijlage.
- **Serial.println(...)**
  met dit commando kan je om het even wat naar de seriële monitor schrijven. Wanneer je bijvoorbeeld de waarde van een variabele wilt zien, dan schrijf je tussen de haakjes de naam van de variabele. Wanneer je bepaalde tekst wilt zien, dan schrijf je die tekst tussen de haakjes met aanhalingstekens errond: "..."
  **Serial.println** staat voor 'iets schrijven en dan naar d elijn erronder gaan'
  **Serial.print** staat voor 'iets schrijven' (zonder een lijn naar onder te gaan).

```
int sensorPin = 5; //sensorPin is het pin-nummer waar de sensor
                     output op zit
                   //de resolutie is 10 mV / °C
                   //500 mV offset om negatieve T mogelijk te
                     maken
void setup()
{
Serial.begin(9600);
}
```

```
void loop()
{
int reading = analogRead(sensorPin); //reading is de variabele
                                       waarin de 10-bit waarde zit
                                       die we in pin A5 uitlezen
float voltage = reading * 5000.0;
voltage /= 1024.0;              // we zetten de variabele reading
                                   om in een output voltage tussen
                                   0mV en 5000mV
Serial.print(voltage);
Serial.println(" mV");          // we printen de output voltage uit
                                   met de eenheid erachter
float temperatureC = (voltage - 500.0) * 0.1 ;
            //De variabele Voltage wordt omgezet in een variabele
              voor temperatuur in °C. De 500 offset is nodig om
              negatieve temperaturen mogelijk te maken. De
              resolutie is 10 mV per °C.
Serial.print(temperatureC);
Serial.println(" degrees C"); //We printen de output voltage uit
                                 met de eenheid erachter
delay(500);      // Een halve seconde wachten vooraleer we de
                    volgende meting doen
}
```

## LUCHTDRUK METEN

### Opdracht

Meet de luchtdruk met de meegeleverde BMP sensor. Laat de gemeten waarden verschijnen in Pa in het seriële monitor venster. Creëer een overdruk of onderdruk rond de sensor om te testen.

### Hardware

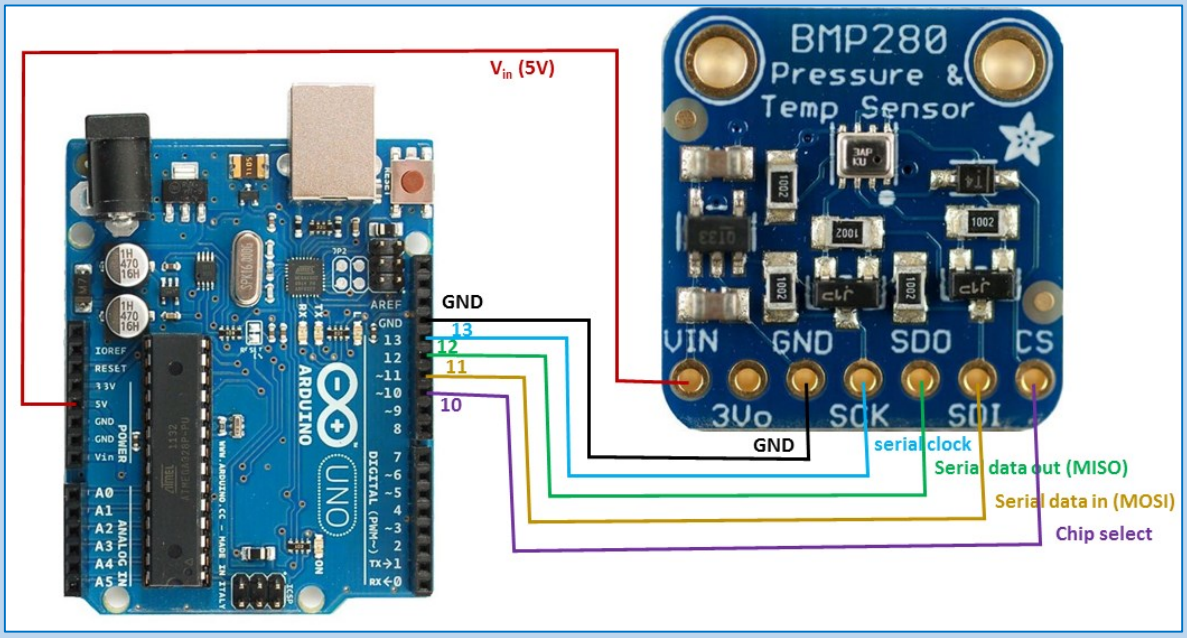
We hebben nodig:

- 1 Arduino Uno
- 1 breadboard + header stacks (6)
- 1 sensor BMP280
- 6 jumper kabeltjes

### Circuit

We sluiten de sensor aan volgens de SPI communicatie interface. Zes sensor pinnen worden als volgt aangesloten:

- Sensor Vin op de 5V Arduino pin
- Sensor GND pin op de Arduino GND pin
- Sensor SCK pin op de Arduino digital 13
- Sensor SDO pin op de Arduino digital 12
- Sensor SDI pin op de Arduino digital 11
- Sensor CS pin op de Arduino digital 10




*Circuit voor de oefening "Luchtdruk meten". Bron: ESERO BE.*

## Sketch

De sketch voor de BMP280 sensor is te vinden op de Adafruit website onder deze link: https://github.com/adafruit/Adafruit_BMP280_Library/archive/master.zip

Je krijgt hier een zip file. Wanneer je deze uitpakt, dan zie je dat er meedere bestanden in zitten:

| | | | |
|---|---|---|---|
| Adafruit_BMP280_Library-master | .github | ISSUE_TEMPLATE.md<br>PULL_REQUEST_TEMPLATE.md | |
| | examples | bmp280test | bmp280test.ino |
| | Adafruit_BMP280.cpp<br>Adafruit_BMP280.h<br>Library.properties<br>README.md | | |

- Een **header file (*.h)** bevat een lijst van definities en bepalingen die in een bibliotheek (library) vereist zijn.
- Een **source file (*.cpp)** bevat de eigenlijke code van een bibliotheek (library).
- Een **sketch (*.ino)** is de sketch zoals je die in het sketch venster van Arduino IDE shrijft.
- Een **Markdown file (*.md)** is een informerend tekstbestand dat je best kan openen met een programma dat HTML leest (uw internet browser dus). Aangezien het een tekstbestand is, kan je het ook openen via Microsoft Notepad of Wordpad of met Apple TextEdit.

De map "Adafruit_BMP280_Library-master" moet je van naam veranderen door het laatste deel weg te laten. De nieuwe naam: "Adafruit_BMP280"

Vervolgens kopieer je de hele map naar :
C:/program files (x86)/Arduino/libraries

Start nu de Arduino IDE terug op (het programma screent op nieuwe libraries alleen maar bij het opstarten).

Vervolgens ga je in het Arduino IDE menu naar:
File > Examples > Adafruit_BMP280 > bmp280test
De sketch om de sensor uit te lezen is nu geopend in het sketch venster:

```
/***************************************************************
  This is a library for the BMP280 humidity, temperature &
  pressure sensor
  Designed specifically to work with the Adafruit BMEP280 Breakout
  ----> http://www.adafruit.com/products/2651
  These sensors use I2C or SPI to communicate, 2 or 4 pins are
  required to interface.

  Adafruit invests time and resources providing this open source
  code, please support Adafruit andopen-source hardware by
  purchasing products from Adafruit!

  Written by Limor Fried & Kevin Townsend for Adafruit Industries.
  BSD license, all text above must be included in any
  redistribution
***************************************************************/

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>

#define BMP_SCK 13
#define BMP_MISO 12
#define BMP_MOSI 11
#define BMP_CS 10

Adafruit_BMP280 bmp; // I2C
//Adafruit_BMP280 bmp(BMP_CS); // hardware SPI
//Adafruit_BMP280 bmp(BMP_CS, BMP_MOSI, BMP_MISO,  BMP_SCK);

void setup() {
  Serial.begin(9600);
  Serial.println(F("BMP280 test"));

  if (!bmp.begin()) {
    Serial.println(F("Could not find a valid BMP280 sensor, check
                                                        wiring!"));
    while (1);
  }
}

void loop() {
    Serial.print(F("Temperature = "));
    Serial.print(bmp.readTemperature());
    Serial.println(" *C");
```

```
    Serial.print(F("Pressure = "));
    Serial.print(bmp.readPressure());
    Serial.println(" Pa");

    Serial.print(F("Approx altitude = "));
    Serial.print(bmp.readAltitude(1013.25)); // this should be
                                             adjusted to your
                                             local forcase
    Serial.println(" m");

    Serial.println();
    delay(2000);
}
```

We komen hier enkele nieuwe commando's tegen die meer uitleg vergen.

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
```

Het commando **"#include**" is een verwijzing naar externe, bestaande bibliotheek hulpbestanden. Merk op: er word geen punt-komma geplaatst achter dit lijntje (anders krijg je foutmelding).

De eerste twee verwijzingen (**Wire.h en SPI.h**) zijn bibliotheken die communicatie in I2C en SPI mogelijk maken.
De volgende twee verwijzingen (**Adafruit_sensor.h en Adafruit_BMP280.h**) zijn bibliotheken eigen aan de sensor.

```
#define BMP_SCK 13
#define BMP_MISO 12
#define BMP_MOSI 11
#define BMP_CS 10
```

Het commando "**#define**" kent een naam toe aan een bepaalde constante. In dit voorbeeld wordt dus in het hele verdere programma de naam BMP_SCK steeds vervangen door de waarde 13.
Het gebruik van #define wordt afgeraden. Wanneer de naam zoals bepaald in #define voorkomt als deel van een woord, dan wordt dit ook vervangen door de waarde. Dit kan onvoorziene fouten veroorzaken.

```
  Serial.println(F("BMP280 test"));
```

Wanneer een F toegevoegd wordt aan een print-opdracht, dan zal het betreffende woord opgeslagen worden in het flash geheugen in plaats van in het RAM geheugen. Op die manier kan men RAM geheugen sparen. Voor de rest verandert de F niets aan het uitschrijven van het woord naar de seriële monitor.

```
  if (!bmp.begin()) {
    Serial.println(F("Could not find a valid BMP280 sensor, check
                                                    wiring!"));
```

```
    while (1);
  }
```

**Bmp.begin()** is een commando die de sensor initialiseert.
Een uitroepteken '**!**' staat voor 'niet' (ontkenning, niet gelijk aan...).
**If (!bmp.begin())** betekent dus: "als de BMP sensor niet geïnitialiseerd wordt" (bijvoorbeeld omdat er geen BMP sensor aangesloten is of omdat hij kapot is).

**While** is een commando die ervoor zorgt dat de loop eeuwig blijft lopen totdat de voorwaarde tussen de haakjes voldaan is.
**While (1)** is een while-loop met een ietwat absurde voorwaarde. Hierdoor blijft deze loop altijd lopen, totdat de gebruiker gaat resetten of de Arduino loskoppelen.

```
    Serial.print(bmp.readTemperature());

    Serial.print(bmp.readPressure());

    Serial.print(bmp.readAltitude(1013.25));
```

De constanten 'bmp.readTemperature' , 'bmp.readPressure' , en 'bmp.readAltitude' zijn constanten die gedefinieerd zijn in de hulpbestanden (header bestanden).

De **temperatuur** is van het data type float (met decimalen na de komma), en wordt gegeven in °C.

De **luchtdruk** is van het data type int (integer, zonder decimalen na de komma), en wordt gegeven in Pa.

De **hoogte** is van het data type int, en wordt gegeven in meter. De berekening van de hoogte is echter ook afhankelijk van de actuele luchtdruk op de grond. Deze moet opgezocht worden en ingegeven worden tussen de haakjes bmp.readAltitude().

## EENVOUDIGE OEFENINGEN :
## Met verkeerslicht gewenste temperatuur aangeven

### Opdracht

< to be completed >

### Hardware

< to be completed >

### circuit

< to be completed >

### sketch

< to be completed >

# DATA OPSLAAN OP SD KAART

## Databestand

De Arduino microchip heeft een intern data-geheugen van 200 bytes. Dit is te klein om voldoende meetgegevens op te slaan.
De meetgegevens moeten daarom best worden weggeschreven naar een geheugenkaart. We doen dit in de vorm van een **tekstbestand**. Dat bestand kan je nadien opslaan als .csv (staat voor 'comma seperated values'). Je doet dit simpelweg door in Windows Verkenner (of Finder van Mac) de extensie .txt te overschrijven met .csv. Een csv bestand kan onder meer gelezen worden door MS Excel.

## Oefening: schrijf een boodschap naar je SD kaart - inleiding

Om data te leren wegschrijven op een SD kaart, gaan we eerst een eenvoudige oefening doen: schrijf de tijd (uur) weg op de kaart. Het wegschrijven van de gemeten data doen we op het einde van deze vorming. De principes zijn uiteraard hetzelfde bij beide oprachten.

**De tijd?**

Hoe sluiten we een SD kaart aan op een Arduino microcontroler? Hiervoor hebben we een extra hulpstuk nodig, namelijk de SD card shield.
Een shield is een extra board die precies past op het Arduino UNO board. Door ze op mekaar te plaatsen, worden alle pins meteen correct verbonden. Op de gebruikte shield in deze vorming zit niet alleen een SD kaarthouder, maar ook een "real time clock" (RTC). We kunnen de seconden van deze RTC gebruiken als data die we op de SD kaart willen opslaan.

## Oefening: schrijf een boodschap naar je SD kaart - Hardware

< to be completed >

## Oefening: schrijf een boodschap naar je SD kaart - Sketch

< to be completed >

# DATA VERZENDEN/ONTVANGEN MET RADIO-COMMUNICATIE: Beperkte inleiding

Het valt buiten het opzet van deze vorming om te werken met radio-communicatie. Om u toch enigszins in te leiden geven we hieronder een klein beetje achtergrond informatie. Het installeren van een radio communicatie via Arduino is echter geen deel van deze cursus, en zal u derhalve hieronder niet vinden. In een latere ESERO vorming zullen we dit wel toevoegen.

## Space link

Een “space link” is het communicatiesysteem tussen de satelliet en het grondstation (of meerdere grondstations). Het bestaat essentieel uit deze twee delen:

- **Telemetrie (TM) downlink**
  Meetgegevens van de payload worden door de satelliet uitgezonden. Deze data worden in het grondstation ontvangen.
- **Telecomand (TC) uplink**
  Besturingscomando’s worden in het grondstation uitgezonden, en door de satelliet ontvangen.

In deze vorming is de TC uplink niets anders dan de besturingsmodule van de drone, bestuurd door de piloot. De TM downlink moeten we zelf voorzien, tenzij we de data alleen maar opslaan in onze satelliet zelf op een SD kaartje (om na landing te recupereren).

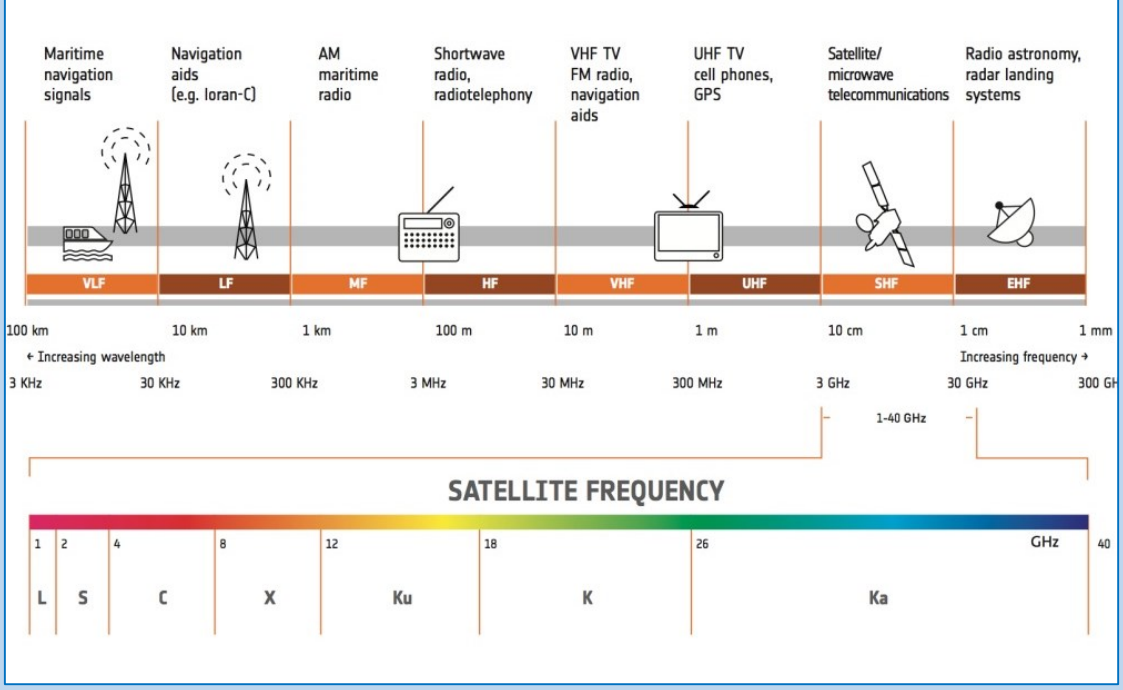
## Radiogolven en frequenties

Radiogolven liggen aan het uiteinde van het elektromagnetisch spectrum. Alle elektromagnetische golven bewegen zich voort aan de lichtsnelheid c. In het luchtledige is c gelijk aan 300.000 km/sec (300.000.000 m/s).

Radiogolven zijn de langste golven. Hun golflengte varieert van 10 cm tot vele kilometers. Zoals bij alle elektromagnetische golven is hun frequentie eenvoudig te berekenen wanneer de golflengte bekend is:

$$\lambda f = c$$

λ = golflengte, f = frequentie, c = lichtsnelheid. Bijgevolg kunnen we afleiden dat de frequenties van radiogolven kleiner of gelijk zijn aan 30.000.000 Hz (30 MHz).
Golven met een kleinere golflengte (en dus grotere frequentie) worden microgolven en radargolven genoemd. De gewone radiogolven ($\lambda \geq 10$ cm) worden op Aarde gebruikt voor radiocommunicatie. Microgolven en radargolven worden gebruikt voor communicatie in de ruimtevaart, omdat ze zonder verstoring door de atmosfeer reizen. Ze worden niet gereflecteerd door de ionosfeer.




*Gebruik van radiogolven voor communicatie. Bron: ESA-ESERO werkgroep Space Robotics.*

## Hoe kan een elektromagnetische golf informatie bevatten?

Stel je een radiogolf voor als een sinusoïde. Het opvangen van zo'n golf in een elektronisch apparaat, zal een variërende spanning veroorzaken. Die variërende spanning kan op elk moment bepaald worden. De spanning die op 1 bepaald moment gemeten kan worden, wordt de 'displacement' genoemd. De displacement y (in Volt) kan je berekenen:

$$Y = A \sin ( 2 \pi f t )$$

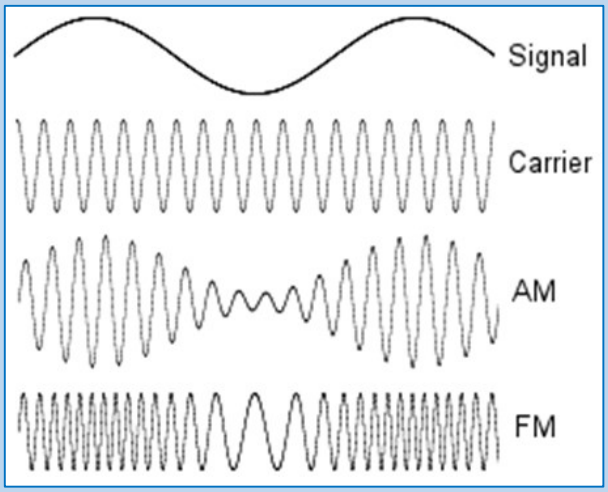
A is de maximale amplitude van de sinusgolf.
f is de frequentie van de golf.
t is het tijdstip van de meting.

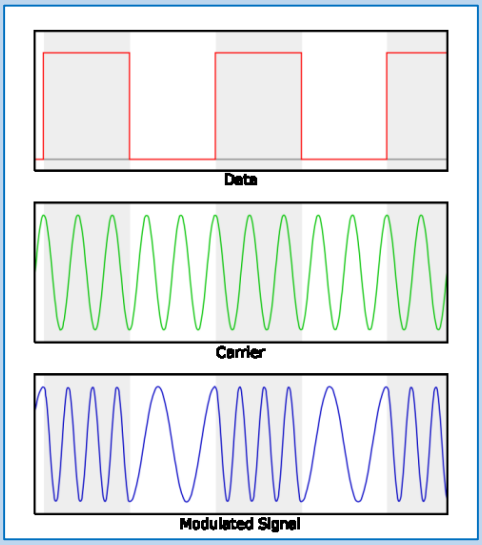
Nu is onze perfecte sinusgolf niet erg interessant om uit te lezen. De displacement zal immers een eentonig patroon vertonen: hoog-laag-hoog-laag-... met een regelmatige frequentie en vaste amplitude. Deze golf is geschikt als **draaggolf (carrier wave)** omdat we kunnen afstemmen op 1 bepaalde frequentie om hem te ontvangen. Maar er moet nog informatie aan toegevoegd worden.

## Modulatie

Bij golfmodulatie gaan we een draaggolf combineren met een golf die de eigenlijk data of boodschap bevat, de **signaalgolf (signal wave)**. Het product van zulke combinatie van draaggolf en signaalgolf is een **gemoduleerde golf**.
De data worden overgebracht in bits (0 of 1, hoog of laag) in een bepaald patroon. De gemoduleerde golf brengt deze bits over met variaties in de amplitude of met variaties in de

frequentie. In het eerste geval spreekt men van Amplitude Modulation of AM. In het tweede geval gaat het over Frequentie Modulation of FM.




*Modulatie van radiogolven.*
*Links: principe van AM en FM.*
*Rechts: een vorm van FM, namelijk binaire 'Frequency Shift Keying', waarbij twee vaste frequenties gebruikt worden voor de O en 1 waarde van de binaire code. Bron: ESA-ESERO werkgroep Space Robotics.*

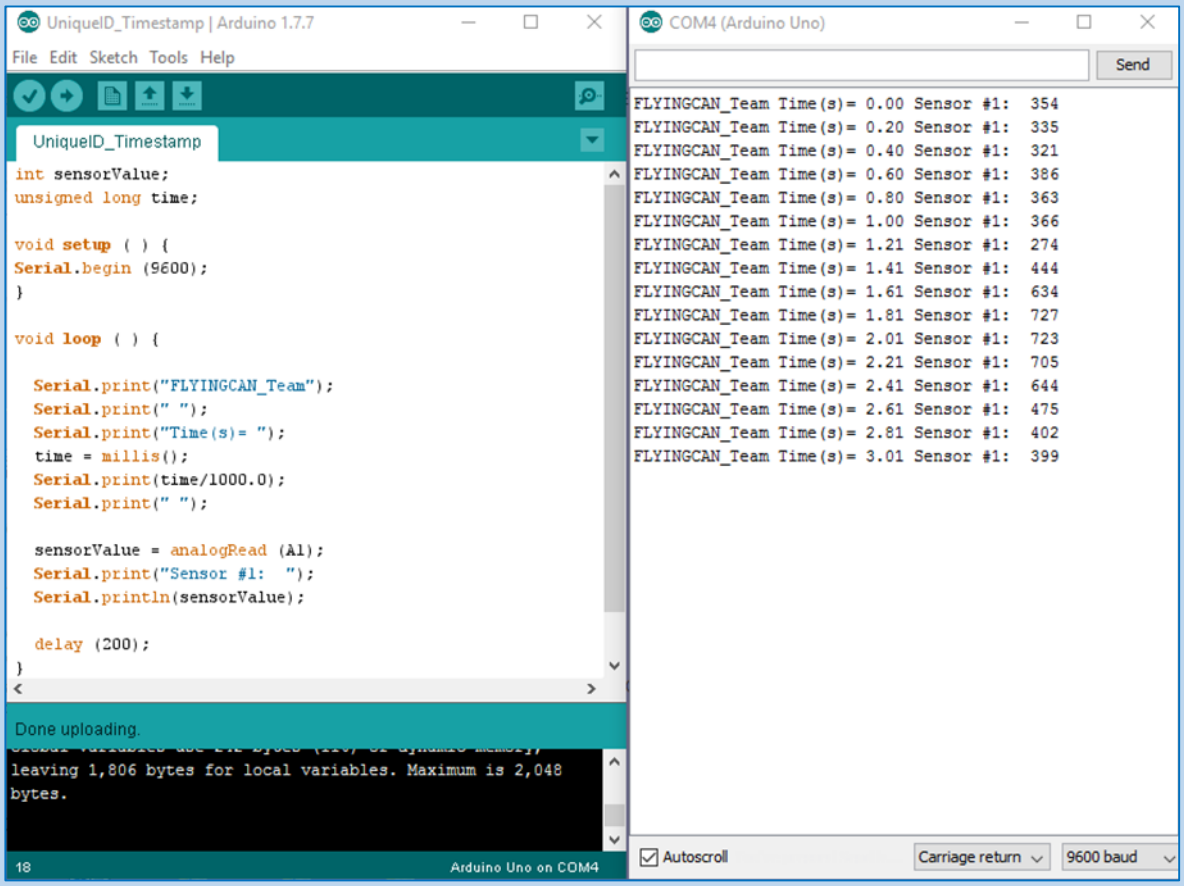
Voor radiocommunicatie met onze mini-satelliet gaan we gebruik maken van **FM**. Deze methode van modulatie heeft voordelen:

- Minder ruis die mee versterkt wordt
- Minder power is nodig om dezelfde informatie te verzenden
- De bandbreedte is groter

## Metadata

**'identifier'**
In principe zal de organisator van een STEM project een unieke frequentie bezorgen die je als team kan gebruiken voor het verzenden en ontvangen van je data. Maar wanneer meerdere teams data per radiocommunicatie versturen, kan het toch nog voorkomen dat er interferentie optreedt tussen meerdere satellietjes. Het is daarom belangrijk om je Arduino ook een unieke code of uniek woord te laten meesturen met de meetgegevens. Zo herken je met zekerheid je eigen data.

**Timestamp**
Bovendien is het ook handig om samen met de meetgegevens de datum en het uur mee te verzenden. Zo geraken data niet gemakkelijk verward.




*Voorbeeld van een sketch waarbij een identifier en een timestamp werd toegevoegd.*
*Bron: ESA-ESERO werkgroep Space Robotics.*

# 2c Je satelliet klaarmaken

## DE AFGEWERKTE SATELLIET: HARDWARE

< to be completed >

## ALLE CODES INTEGREREN IN 1 SKETCH

### Structuur van de volledige sketch

Om deze sketch volledig te begrijpen, moeten we eerst de structuur ervan begrijpen. De structuur is als volgt:

1) verwijzingen naar gebruikte **bibliotheken**
2) Definiëren van input/output functies van bepaalde **pins**

**SET UP**

3) **Communicatiesnelheid** bepalen tussen microcontroller en seriële monitor.

4) **Intitialiseren** van **sensor** en **SD kaart** : checken of deze componenten door de microcontroller herkend worden als functionerend.
5) Open het **datalog bestand** waar we alle data in gaan schrijven (datalog.txt).
6) **Initialiseren** van de real time **clock** : functionalteit van de klok checken en begin datum/uur vastleggen.

### Herhalende LOOP

7) We schrijven de huidige **tijd** (jaar, maand, dag, uur, min, sec) naar het datalog bestand.
8) We schrijven de huidige **temperatuur** (°C) naar het datalog bestand.
9) We schrijven de huidige **hoogte** (m) naar het datalog bestand.
10) We gaan naar de **volgende lijn** in het datalog bestand.
11) We schrijven de huidige **tijd** (jaar, maand, dag, uur, min, sec) naar de seriële poort.
12) We schrijven de huidige **temperatuur** (°C) naar de seriële poort.
13) We schrijven de huidige **hoogte** (m) naar de seriële poort.
14) We gaan naar de **volgende lijn** in de seriële poort.
15) We schrijven de nieuwe data (tijd, temp, hoogte) naar de **SD kaart.**
16) We wachten een **halve seconde.**

### Code van de volledige sketch

De onderstaande sketch moeten we volledig kopiëren in de Arduino IDE, en opladen naar de microcontroller.

```
// Eerst doen we de verwijzingen (#include) naar bibliotheken
   die we gedownload hebben op onze computer. De bibliotheken
   werden opgeslagen onder
   C:/program files(x86)/Arduino/libraries
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include "RTClib.h"
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>

// vervolgens geven we een aantal pins een input of output
   functie (sensor communicatie).
#define BMP_SCK 13
#define BMP_MISO 12
#define BMP_MOSI 11
#define BMP_CS 10

Adafruit_BMP280 bmp;
// I2C
// SD variables
const int chipSelect = 4; // we leggen de waarde van variabele
                          chipSelect vast op 4 (niet meer
                          veranderbaar).
File dataFile;
// Real Time Clock (RTC) variabelen
RTC_PCF8523 rtc;
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday",
   "Wednesday", "Thursday", "Friday", "Saturday"};
```

```
void setup()
{
  Serial.begin(57600); // Dit is de baud snelheid die we nodig
                          hebben om de RTC correct te laten
                          verlopen.
    if (!bmp.begin()) {  // controleren of de sensor BMP280
                          herkend wordt als functionerende
                          sensor.
    Serial.println(F("Could not find a valid BMP280 sensor,
    check wiring!"));
    while (1);           // doe niets meer tot reset wegens
                          fout:
  }

  Serial.print("Initializing SD card...");
  pinMode(CS, OUTPUT); // Zorg ervoor dat de chip select pin
                          geschakeld is als uitgang, zelfs al
                          gebruik je ze niet
  if (!SD.begin(chipSelect)) {  // controleren of de SD kaart
                                   kan geopend worden via pin4.
    Serial.println("Card failed, or not present");
    while (1) ;                 // doe niets meer tot reset
                                   wegens fout:
  }
  Serial.println("card initialized.");

  dataFile = SD.open("datalog.txt", FILE_WRITE);
                        // Open een nieuw bestand waar we de
                        data in gaan loggen.
  if (! dataFile) {     // controleren of dataFile nu bestaat.
    Serial.println("error opening datalog.txt");

    while (1) ;         // doe niets meer tot reset wegens
                          fout:
  }

// RTC initializatie
  if (! rtc.begin()) {  // controleren of de real time clock
                          functioneert en kan herkend worden.
    Serial.println("Couldn't find RTC");

    while (1);          // doe niets meer wegens fout:
  }

  if (! rtc.initialized()) {   // controleren of de tijd aan
                                  het lopen is.
    Serial.println("RTC is NOT running");
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // Zet de datum en tijd in de RTC chip naar het moment
      waarop het programma is gecompileerd. Als je liever zelf
      een tijd en datum kiest zoals bv: 21 januari, 2014 om 3u
      dan gebruik je volgend commando:
      rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
  }
```

```
void loop()
{
  // We schrijven de actuele tijd naar het databestand:
  DateTime now = rtc.now();
  dataFile.print(now.year(), DEC);
  dataFile.print('/');
  dataFile.print(now.month(), DEC);
  dataFile.print('/');
  dataFile.print(now.day(), DEC);
  dataFile.print(' ');
  dataFile.print(now.hour(), DEC);
  dataFile.print(':');
  dataFile.print(now.minute(), DEC);
  dataFile.print(':');
  dataFile.print(now.second(), DEC);
  dataFile.print(';');
  // we schrijven de actuele temperatuur naar het databestand:
  dataFile.print(bmp.readTemperature());
  dataFile.print(';');
  // we schrijven de actuele hoogte naar het databestand:
  dataFile.println(bmp.readAltitude(1021));
  // het getal tussen haakjes dient aangepast te worden aan de
   huidige waarde van de luchtdruk (in hPa) om een juiste
   berekening van de hoogte te bekomen. Deze berekening is
   opgenomen in de BMP280 bibliotheek waarnaar we verwezen
   hebben.

  // Hieronder doen we hetzelfde opnieuw, maar schrijven we
     naar de seriele poort:
  Serial.print(now.year(), DEC);
  Serial.print('/');
  Serial.print(now.month(), DEC);
  Serial.print('/');
  Serial.print(now.day(), DEC);
  Serial.print(' ');
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.print(':');
  Serial.print(now.second(), DEC);
  Serial.print(';');
  Serial.print(bmp.readTemperature());
  Serial.print(';');
  Serial.println(bmp.readAltitude(1021));
  // The following line will 'save' the file to the SD card
     after every line of data - this will use more power and
     slow down how much data you can read but it's safer! If
     you want to speed up the system, remove the call to
     flush() and it will save the file only every 512 bytes -
     every time a sector on the SD card is filled with data.
  dataFile.flush();  // Bewaar het bestand op de SD kaart na
                        elke datalijn die eraan toegevoegd
                        werd.
```

```
    delay(500); // wacht een halve seconde, en herhaal dan de
                   loop
    }
```

## Details van deze sketch

Hieronder geven we wat meer uitleg bij bepaalde uitdrukkingen:

**const**

```
    const int chipSelect = 4;
```

Met het commando "**const**" kan je een bepaalde variabele een vaste waarde geven. Eens die waarde met dit commando bepaald is, dan kan je die in de hele verdere sketch niet meer veranderen. Het is een 'read-only' waarde geworden.

Let op: bij het 'const' commando wordt een 'is gelijk' teken gebruikt (=), en de uitdrukking wordt beëindigd met een punt-komma (;). Dit is niet het geval bij het commando '#define'. Dit laatste kan ook gebruikt worden om een bepaalde waarde te geven aan een variabele, maar deze waarde kan nadien wel nog gewijzigd worden.
Bovenstaande uitdrukking kan dus als alternatief dit worden:

```
#define int chipSelect 4
```

In de uitdrukking uit onze sketch (**`const int chipSelect = 4;`**) krijgt de variabele 'chipSelect' de constante invulling 4. Waar we in de verdere code de variabele chipSelect gebruiken, staat dit dus altijd gelijk aan de waarde 4. De variabele een andere waarde geven, zou een foutmelding opleveren.

**!SD.begin**

```
    if (!SD.begin(chipSelect)) {
```

Het uitroepteken staat voor een ontkenning ('niet').
Bijvoorbeeld, de uitdrukking: **`If (!x < 0)`** betekent letterlijk: "als x niet kleiner is dan nul".

In de shield die we gebruiken (waarop het SD kaart slot zit), wordt pin 4 gebruikt als de chip select pin (CS) van de SD kaart. Dat is als het ware het 'adres' van de SD kaart.
Herinner u dat de waarde van de variabele 'chipSelect' werd vastgelegd op 4. Je moet de variabele in de uitdrukking hierboven dan ook gewoon zien als de waarde 4. Er staat dus eigenlijk:

```
if (!SD.begin(4))
```

Het woord '**begin**' staat voor het openen of opstarten van de SD kaart. De uitdrukking betekent letterlijk: "Als de SD op pin 4 niet opent ...".

**While (1)**

```
    if (!bmp.begin()) {
      Serial.println(F("Could not find a valid BMP280 sensor,
                       check wiring!"));
      while (1);
    }
```

While loops blijven eeuwig herhalen totdat de stelling tussen haakjes niet meer klopt.

Bijvoorbeeld:

```
While (OurVariable < 200)
{
      doSomething;
      OurVariable++;
}
```

Bovenstaande uitdrukking betekent:

Zolang de variabele kleiner is dan 200 doe je telkens opnnieuw het volgende:

a) Do domething
b) Tel +1 op bij de variabele

Nu is de uitdrukking “while (1)” wel een beetje eigenaardig. De voorwaarde om de loop te herhalen is gewoon “1”. Dit lijkt niet echt op een zinvolle voorwaarde.
Deze uitdrukking wordt gebruikt wanneer een loop telkens weer herhaald moet worden totdat ergens een beginsituatie verandert. Je zou kunnen zeggen: 1 is en blijft altijd 1. En daarom is de voorwaarde voor “while(1)” altijd voldaan. Dus de loop blijft zich gewoon herhalen.

In onze sketch wordt de uitdrukking “while(1)” gebruikt in combinatie met “if”. De voorwaarde onder deze “if” gaat hier over het herkennen van een functionerende sensor. Wanneer deze sensor niet gevonden wordt, dan blijft de loop zich herhalen (het schrijven van de boodschap ‘could not find...’ naar de seriële monitor) totdat de gebruiker het programma onderbreekt, en de hardware aanpast.

**dataFile**

```
      dataFile = SD.open("datalog.txt", FILE_WRITE);
```

We moeten een tekstbestand op de SD kaart aanmaken waarin we de data kunnen opslaan. We zullen dit bestand in deze oefening de naam datalog.txt geven, maar het kan om het even welke naam zijn.

In de verdere code zullen we naar dit bestand verwijzen met “`dataFile`”.

Het commando `SD.open("fileName",FILE_WRITE)` zal een nieuw bestand aanmaken (tenzij er al een bestand met deze naam bestaat) op de SD kaart. Het achtervoegsel `FILE_WRITE` zorgt ervoor dat we read and write toegang hebben tot het nieuwe bestand. Het alternatief is `FILE_READ`, wanneer we een bestand willen openen alleen om uit te lezen.

De bestandsnaam tussen aanhalingstekens (`"datalog.txt"`) kan ook een een mappenstructuur bevatten. Bijvoorbeeld: `"ESEROvorming/20171109/datalog.txt"`. Zonder deze mappenverwijzing wordt het nieuwe bestand gewoon op de hoofddirectory van de SD kaart geschreven.

**Flush**

```
    dataFile.flush();
```

Het commando `Flush` zorgt ervoor dat na elke nieuwe lijn met toegevoegde data het databestand overschreven wordt op de SD kaart. Hierdoor wordt meer stroom verbruikt, en wordt de snelheid van het uitlezen van data vertraagd. Maar het is veiliger.

Als je het `Flush` commando verwijdert, dan zal het databestand telkens overschreven worden nadat er 512 bytes nieuwe data toegevoegd werden. 512 bytes is immers het maximum aan data dat de microcontroller intern kan opslaan. Het uitlezen van data kan dan sneller gaan (en met minder stroomverbruik), maar er is een hoger risico om data kwijt te raken wanneer er iets misgaat.

## SOLDEREN

## WAAROM SOLDEREN ?

Het voordeel van het vastsolderen van satellietonderdelen is heel duidelijk: ze komen niet los tijdens de lancering of de vlucht. Een goed gesoldeerde component is super betrouwbaar.
Is het solderen noodzakelijk voor elk elektronisch aangestuurd experiment. Hierop is maar 1 mogelijk antwoord: JA, het is absoluut noodzakelijk. Het risico dat al je werk verloren gaat door een loskomend contact is heel groot als ze niet gesoldeerd zijn.

## VEILIGHEID

- Een soldeerbout heeft een operationele temperatuur van 300 à 400°C. Wees dus erg voorzichtig met de soldeertip om brandwonden te vermijden.
- Zet de soldeerbout altijd uit de buurt van andere voorwerpen.
- Schakel de soldeerbout uit zodra het werk gedaan is, en laat het voldoende afkoelen voordat je het opbragt in een kast.
- Draag een veiligheidsbril.

## MATERIAAL

### Soldeermetaal

Met soldeermetaal bedoelen we de metaaldraad die je laat smelten op de plek van de aan te maken verbinding.
Het traditionele soldeermetaal is een loodlegering. Dit is een legering die bij het smelten heel mooi en vlot in het gaatje vloeit. Het is daarom het gemakkelijkste soldeermetaal voor beginners. Maar vanwege de loodhoudende damp bij het smelten, wordt het gebruik op school door sommigen afgeraden. Strikt genomen is het zelfs verboden op school.

### Soldeerstation

De soldeerbout is het centrale gereedschap bij het solderen. Het heeft een metalen punt die opwarmt tot ongeveer 400°C. Gewoonlijk wordt het geleverd met een handige houder, die toelaat dat je de hete soldeerbout uit je handen zet zonder gevaar.

De kostprijs van een goedkoop soldeerstation ligt in de grootte-orde van 25 euro. Je kan er prima mee solderen. Het voordeel van duurdere toestellen is dat ze gewoonlijk veel langer intact blijven, zelfs al laat je ze dagelijks continu aanstaan.

### Accessoires

Je houdt best bij de hand:

- Een kleine tang met puntig uiteinde
- Een kniptang om kabeltjes af te knippen
- Een striptang om kabeltjes van isolatie te ontdoen
- Eventueel: een vergrootglas op een voet (handenvrij)
- Goede verlichting

## VOORBEREIDING

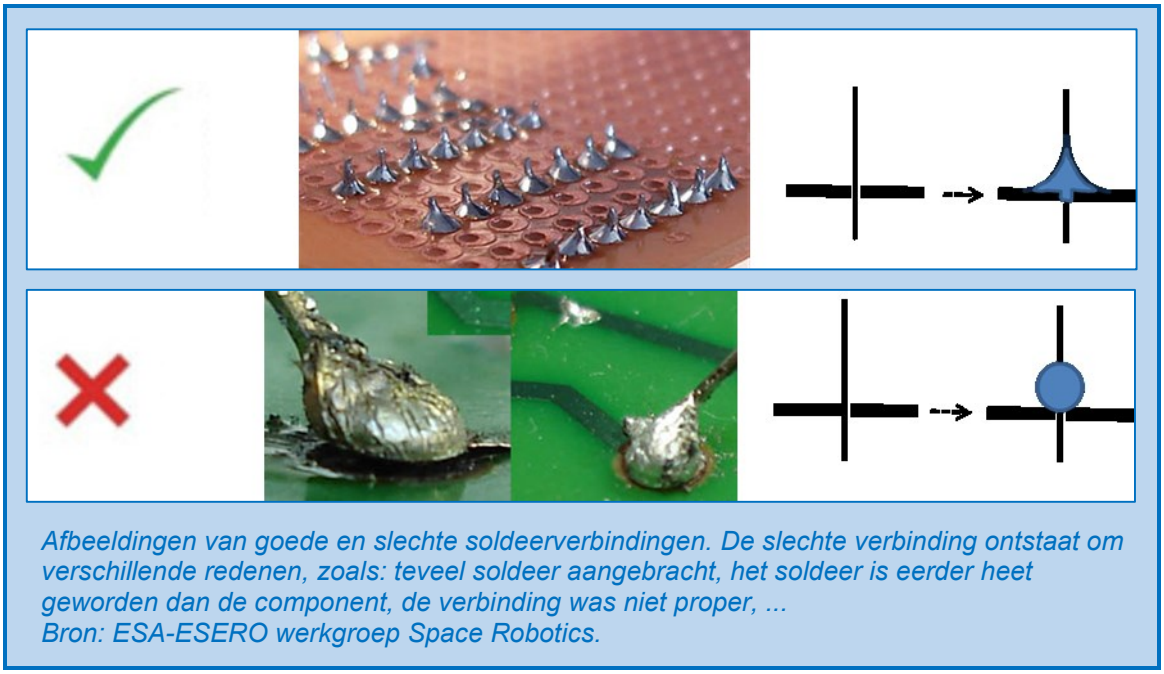
1. Plaats de soldeerbout in zijn standaard en stop de stekker in het stopcontact. De soldeerbout heeft een paar minuten nodig om tot de **juiste temperatuur** van ongeveer 350 - 400°C te komen.
2. Bevochtig de **spons** in de standaard. De spons moet vochtig zijn maar niet kletsnat. Knijp het teveel aan water eruit.
3. Wacht enkele minuten tot de soldeerbout **op werktemperatuur** is. Je kan dit checken door een klein beetje soldeer tegen de punt te houden. Smelt het goed dan is de soldeerbout op de juiste temperatuur.
4. Nu de soldeerbout op temperatuur is veeg je **de punt schoon** op de vochtige spons
5. **Smelt een klein beetje soldeer** op de soldeertip. Dit wordt vertinnen genoemd en het helpt om de hitte van de punt te verplaatsen naar de te maken verbinding. Dit doe je alleen nadat je de punt hebt schoongemaakt op de spons.

## SOLDEREN : INSTRUCTIES

Je bent nu klaar om te gaan solderen.

1. Hou de soldeerbout vast **als een pen**. Doe net alsof je je naam gaat schrijven. Kijk uit dat je het hete gedeelte niet aanraakt.
2. **Raak de plek** waar je de verbinding wil gaan maken aan **met de soldeerpunt**. Zorg ervoor dat de punt zowel het onderdeel als de printbaan of het andere onderdeel aanraakt. Hou de soldeerpunt op deze plek voor 3 seconden en dan....
3. **Plaats wat soldeer** op het te solderen punt. Als alles goed op tempratuur is zal het soldeer soepel vloeien over de printbaan en het component of componenten onderling. Het moet de vorm van een vulkaan aannemen. Voeg soldeer toe tegen de te maken verbinding en niet tegen de soldeerbout.

4. **Haal de soldeer weg** en daarna de soldeerbout. Laat de gemaakte verbinding even afkoelen en houdt de verbinding stil !
5. **Inspecteer** de verbinding goed! Het soldeer moet glimmen en de vorm hebben van een vulkaan. Als dit niet zo is dan moet je de verbinding opnieuw verwarmen en wat meer soldeer toevoegen. Zorg er dit keer voor dat zowel de printbaan als het onderdeel goed heet zijn.



*Afbeeldingen van goede en slechte soldeerverbindingen. De slechte verbinding ontstaat om verschillende redenen, zoals: teveel soldeer aangebracht, het soldeer is eerder heet geworden dan de component, de verbinding was niet proper, ...*
*Bron: ESA-ESERO werkgroep Space Robotics.*

Solderen kan in principe vanaf twee verschillende zijden:

- **Thrue hole soldering**
  Het component pootje/pinnetje wordt langs de ene kant van de printplaat in een gaatje gebracht, het soldeer wordt erin gesmolten vanaf de andere kant (onderkant).
- **Smt**
  Alles langs dezelfde kant. Met een gewone soldeerbout is deze techniek wel iets moeilijker.

Een checklist voor een perfecte soldeerverbinding

1. Alle delen moeten goed schoon zijn en niet geoxideerd.
2. Zet de te solderen printplaat vast in een standaard zoals een helping hand of een andere manier.
3. Vertin de hete soldeerpunt met een kleine hoeveelheid soldeer. Doe dit ook altijd met nieuwe soldeerpunten die voor het eerst worden gebruikt.
4. Maak het uiteinde van de hete soldeerpunt op een vochtige spons schoon.
5. Verwarm beide delen van de verbinding met de soldeerbout voor ongeveer een seconde.
6. Blijf verwarmen en voeg vervolgens voldoende soldeer toe om een adequate verbinding te vormen.
7. Verwijder de soldeerbout en zet deze terug in de soldeerstandaard.
8. Het vergt totaal slechts twee of drie seconden hoogstens, om de gemiddelde verbinding te solderen.

9. Beweeg geen onderdelen tot het soldeer is afgekoeld en de soldeerverbinding hard is.
10. Controleer of de verbinding goed is gemaakt.

Meer info:
https://learn.sparkfun.com/tutorials/how-to-solder-through-hole-soldering

# 3 Lancering

**Drone vlucht**
Harnas, bevestiging
**Klein woordje over lancering bij ABC**

# 4 Data verwerken en presenteren

## Kalibratie en verwerking: inleiding

Je kan de Arduino zo programmeren dat de ruwe data (meetgegevens) reeds in de payload gekalibreert en verwerkt worden. Op die manier kan je in het grondstation meteen de gewenste metingen aflezen zoals je ze wou hebben. Tijdens de ESERO vorming zullen we dit doen om tijd te winnen.

Maar dit wordt afgeraden. Het is beter de ruwe data eerst te verwerven, en ze later met de computer te verwerken. Zo is het risico dat er iets fout gaat veel lager. Bovendien zijn de ruwe data erg waardevol wanneer je vreemde resultaten gekregen hebt, en je wilt de verwerking opnnieuw doen, of op een andere manier. Veel leerlingenteams krijgen bij hun eerste elektronisch aangestuurde experimentjes een set van (op het eerste zicht) zinloze of absurde data terug. In dat geval is er nog hoop wanneer de ruwe data opnieuw kunnen verwerkt worden.

## TEMPERATUUR

Omdat we in de ESERO vorming gebruik maken van de BMP280 sensor, zullen we meteen de verwerkte temperatuur, luchtdruk en hoogte krijgen op de SD kaart. Desondanks wordt hieronder toch uitgelegd hoe de kalibratie van temperatuur kan gebeuren wanneer je een analoge sesnsor gebruikt. Hoewel dit in de ESERO vorming niet nodig is, kan het toch erg nuttig zijn wanneer u zelf een electronische meting wil doen op school.

### Kalibratie van thermistor output

De output data van een simpele thermistor zijn variabele spanningen, met Volt als eenheid. Je kan deze omzetten in corresponderende temperaturen in graden Celsius.

Voor de meest voorkomende temperaturen op Aarde, kunnen we ervan uitgaan dat de thermistor output (gemeten spanning) en de temperatuur een lineair verband hebben. Het volstaat daarom enkele testjes vooraf te doen waarbij de temperatuur bekend is (gemeten met thermometer), en de corresponderende spanning over de thermistor gemeten wordt. Wanneer je deze punten uitzet op een grafiek, dan kan je de best passende rechte vinden die de relatie tussen outputspanning en temperatuur vervolledigt.

Nu willen we natuurlijk een formule, liever dan een lijn op een grafiek, om de gemeten spanningen om te zetten in temperaturen. De algemene wiskundige formule van een lineair verband (een schuine rechte) is als volgt:

$$y = mx + c$$

De waarde m bepaalt de helling van de rechte.
De waarde c bepaalt waar de lijn zijn nulpunt heeft (de x-as doorsnijdt).
Toegepast op de lineaire verhouding tussen thermistor spanning en temperatuur kan je stellen:
Y = temperatuur
X = thermistor spanning

In onze grafiek die gebaseerd is op enkele testmetingen, kunnen we de waarde m vinden als volgt:

$$m = \Delta y / \Delta x$$

Het volstaat nu om de Δy en Δx van de uiterste testwaarden in te vullen, zodat de helling m bekend is:

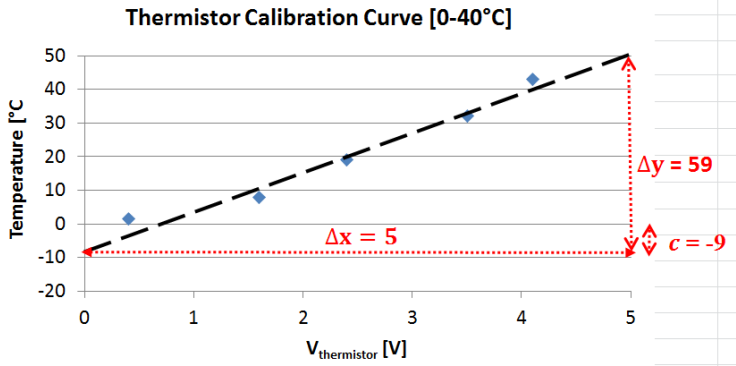
| $V_{thermistor}$ [V] | Temperature [°C] |
|---|---|
| 0.4 | 1.5 |
| 1.6 | 8 |
| 2.4 | 19 |
| 3.5 | 32 |
| 4.1 | 43 |

$y = mx + c$

$m = \frac{\Delta y}{\Delta x} = \frac{59}{5} = 11.8$

$c = -9$

$Temperature = 11.8(V_{thermistor})$ - 9




*Kalibratie van thermistor waarden tot corresponderende temperatuur. Voorbeeld van 5 gemeten testwaarden, corresponderende grafiek en kalibratieformule.*
*Bron: ESA-ESERO werkgroep Space Robotics.*

## LUCHTDRUK EN HOOGTE

### De hoogte berekenen op basis van luchtdrukwaarden

De relatie tussen atmosferische druk en temperatuur wordt gegeven door de formule:

$$P = 101325\ (1 - 2.25577\ 10^{-5}\ h)^{5.25588}$$

Hierbij is de luchtdruk P uitgedrukt in Pascal (Pa), en de hoogte h in meter boven zeeniveau (m). De waarde 101325 is een gemiddelde luchtdruk op zeeniveau. Je dient deze waarde aan te passen aan de luchtdruk op zeeniveau op het moment van uw vlucht. Deze waarde kan je vinden op de website van de meteorologische dienst van uw land.

Om de hoogte te vinden wanneer enkel de luchtdruk bekend is (we meten immers luchtdruk met onze sensor), wordt de formule als volgt herschreven:

$$h = \frac{10^{\left(\frac{log\left(\frac{P}{101325}\right)}{5.25588}\right)} - 1}{-2.25577 \times 10^{-5}}$$

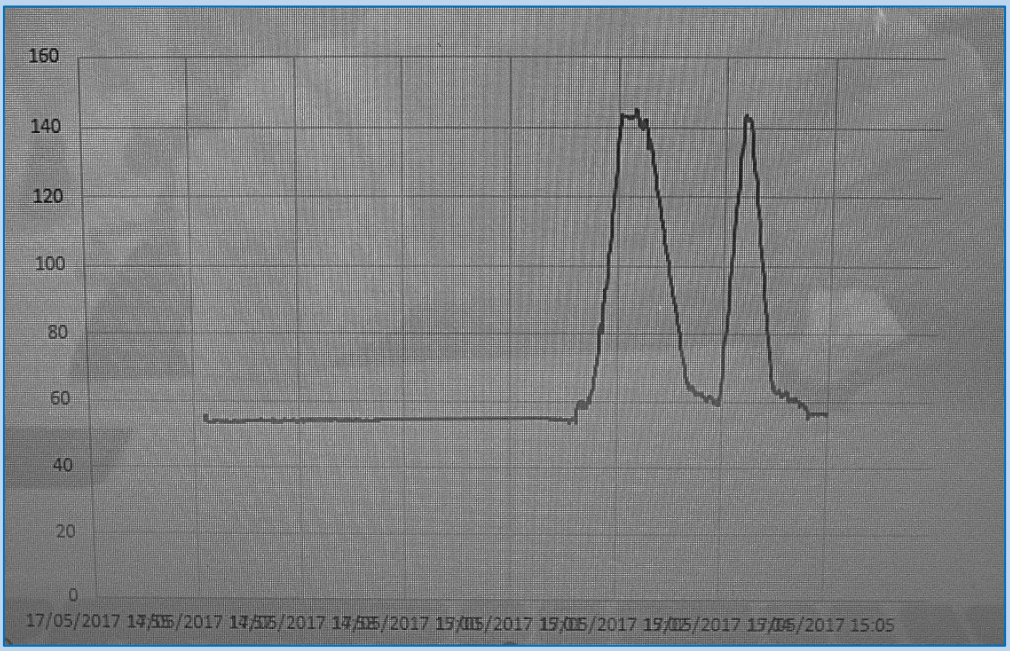
Zo kan je de hoogte van de vlucht per seconde reconstrueren. Alweer: vergeet niet de waarde 101325 aan te passen aan de luchtdruk op zeeniveau voor jou locatie en tijdstip. Een niet aangepaste waarde kan een fout in berekende hoogte opleveren van meer dan 10 meter.

## De gegevens verwerken van de BM280

Let op: ook hier moet vooraf in de sketch de actuele luchtdruk op zeeniveau aangepast zijn aan uw locatie en tijdstip. In de sketch is verder alles voorzien zodat het output .txt bestand kan gelezen worden in excel. Het bevat per tijdstip (meerdere per seconde) metingen van de temperatuur en hoogtes (die laatste berekend op basis van de gemeten luchtdruk).

Op de SD kaart vinden we na de vlucht een txt bestand. Ga als volgt te werk:

- Sla het txt bestand op in uw computer
- Ga naar Windows Verkenner, en zoek het txt bestand
- Klik twee keer op het bestand: je krijgt de mogelijkheid om de naam van het bestand te veranderen
- Verander niet alleen de naam (naar keuze), maar verander ook de extensie .txt in .csv (comma separated values).
- Open dit bestand nu in MS Excel
- Nu heb je alle gegevens in een gemakkelijk te verwerken database. Pas eventueel de layout aan, zodat de data overzichtelijk weergegeven worden.
- Maak grafiek




*De verwerkte meetgegevens van uw mini-satelliet in een tabel en een grafiek. Bron: ESERO BE. <to be replaced by Excel graph>*

# Bronnen


- Lerarenvorming door FabLab Klein-Brabant april-mei 2017 (Davy Vanden Bergh)
- Lerarenvorming door Collège Saint-Michel mei-juni 2017 (Nicolas de Generet)
- Cursus CanSat primaire missie van ESA Education (working group ESA/ESERO): draft version June 2017.
- Teacher training Primary CanSat mission in Europe Jan 2015 (T-Minus).
- YouTube: Arduino voor beginners: Ohm my God https://www.youtube.com/watch?v=zWIsXmtp3Ow&list=PL3ZWCJtjleYHCB0cpVPtunimiERmMy3I5&index=3#t=780.552917
- Evans, Brian W. (2007). Arduino programming notebook. http://creativecommons.org/licenses/by-nc-sa/3.0/
- Adafruit learning resources: https://learn.adafruit.com

# Bijlagen

## Data types

| Data type | RAM geheugen | | Grenswaarden |
|---|---|---|---|
| boolean | 1 byte | | 0 to 1 (True or False) |
| byte | 1 byte = 8 bits | Numerieke waarden zonder decimalen na de komma | 0 to 255 |
| char | 1 byte | | -128 to 127 |
| unsigned char | 1 byte | | 0 to 255 |
| int | 2 bytes = 16 bits | Numerieke waarden zonder decimalen na de komma | -32,768 to 32,767 |
| unsigned int | 2 byte | | 0 to 65,535 |
| word | 2 byte | | 0 to 65,535 |
| long | 4 bytes = 32 bits | Numerieke waarden zonder decimalen na de komma | -2,147,483,648 to 2,147,483,647 |
| unsigned long | 4 byte | | 0 to 4,294,967,295 |
| float | 4 bytes = 32 bits | Numerieke waarden met decimalen na de komma | -3.4028235E+38 to 3.4028235E+38 |
| double | 4 byte | | -3.4028235E+38 to 3.4028235E+38 |
| string | 1 byte + x | | Arrays of chars |
| array | 1 byte + x | Waarden die elk een index nummer dragen. Ze worden opgeroepen via dit indexnummers. De waarde voor elk index nummer moet vooraf bepaald worden door de gebruiker. | Collection of variables |