Arduino cursus 2018

voor beginners en Gevorderden

Naar een publicatie van Paul van de Veen en geschikt gemaakt voor ICT-onderwijs.

Inhoudsopgave

0 Gebruik van deze cursus.

Deze cursus is bedoeld om projecten na te doen en dient ook als voorbeeld hoe je componenten met de arduino kunt laten samenwerken.
Deze cursus is 4 stukken op te delen;

1. Basistechnieken
2. Programma's met libraries
3. Meten met de arduino
4. Input met verschillende devices

Daarnaast is er ook een WIKI met een cursus hoe je een arduino kunt programmeren.
https://maken.wikiwijs.nl/124483/Arduino___Programmeren_voor_beginners

De bedoeling is dat je straks in staat bent om een probleem te analyseren en de arduino zodanig te programmeren dat de arduino kunt inzetten om het probleem werkend te krijgen.

Veel plezier bij deze cursus.

Luuk Ottjes

1 Voorwoord

Arduino cursussen zijn er genoeg. Maar de meeste cursussen gaan niet veel verder dan wat knipperende Led's, enkele schakelaars, een buzzertje en een lichtsensor. Soms wordt er één klein dc-motortje meegeleverd. Soms een enkele stappenmotor. Vaak zit er ook een lcd-display bij dat met een heleboel draadjes aangesloten moet worden.
Zo'n cursus is leuk om een idee te krijgen van wat met Arduino zóu kunnen. Maar meer ook niet. Met één klein zwak motortje bouw je geen autootje dat obstakels kan vermijden. Met één zwakke stappenmotor kun je eigenlijk niets. En dat display moet met zóveel verbindingen via het breadboard aangesloten dat er altijd wel iets los zit. En anders zijn alle uitgangspinnen 'op'.
Deze Arduino cursus probeert dat anders te doen. Bij deze cursus worden onderdelen meegeleverd waarmee je wel echt iets kunt. Na een paar inleidende projecten wordt in plaats van een klein buzzertje een audioversterker toegepast. Sterk genoeg voor spectaculair veel kabaal!
En in plaats van een paar losse led's worden Neopixel led's gebruikt. Dat geeft spectaculaire lichteffecten.
Een display kan veel handiger aangesloten worden door gebruik te maken van het idee van een 'bus'. Dat is iets dat onontkoombaar, onvermijdelijk en superhandig is. In plaats van een heleboel verbindingen naar sensoren, schakelaars, led's, een display enz. gebruiken we iets intelligentere onderdelen die we als één lange lijn achter elkaar kunnen plaatsen. Elk onderdeel in zo'n lijn heeft zijn eigen 'adres' en kan apart ondervraagd worden naar de waarde van de sensor of schakelaar, of kan op dat adres z'n eigen opdrachten krijgen. Met een busstructuur neemt de betrouwbaarheid van alles enorm toe en dus kunnen er ingewikkelder projecten worden gebouwd.
Een ander punt is dat in veel Arduino cursussen gebruik wordt gemaakt van "*delay's*" om de tijd dat een led aan of uit staat te programmeren. Of om te programmeren dat om de zoveel seconden een sensor moet worden gelezen. *Delays are evil!* Al die tijd doet Arduino niets. Dat kan beter. In de sketches waar iets gemeten wordt, wordt die techniek steeds gebruikt.
Led's die alle kleuren kunnen weergeven zijn leuk om mee te werken. Dat begint eenvoudig maar eindigt met het bedienen van een RGB Led via een app op een Android smartphone.
Kleine schakelaars zijn in de meeste cursussen de enige vorm van bedieningselementen. Maar zodra er veel schakelaars nodig zijn wordt dat met de bedrading veel te ingewikkeld. Hier worden twee soorten keypad's gebruikt waardoor er niet 1 of 2 maar gelijk 12 schakelaars beschikbaar komen. Zonder gedoe met draden. In combinatie met de bewegingssensor komt dan een alarmsysteem met 4-cijferige pincode ineens binnen handbereik.
Bij zulke uitbreidingen en combinaties zal al snel ook behoefte zijn om in de Arduino gegevens op te slaan. Dat kan, eenvoudig zelfs, door op te slaan in EEPROM.
Alle componenten zijn geselecteerd op lage kostprijs, op gebruiksgemak en op leesbaarheid van de toegepaste code. Want het is de bedoeling dat deze cursus inspireert om te varieren en te combineren. Arduino leer je door projecten te bedenken en vervolgens uit te puzzelen hoe dat gerealiseerd zou kunnen worden.

Have fun with Arduino!

Paul van de Veen

Introductie

▪ Wat is een Arduino?

Arduino is de naam van een microprocessorbord. Een microprocessor is een kleine computerchip die geprogrammeerd kan worden. Je kunt met een Arduino allerlei apparaten maken. Een robotje, een elektronische dobbelsteen, een looplicht, een alarmsysteem, enz.
Er zijn veel verschillende Arduino's. Sommige groot, andere veel kleiner. De eigenlijke chip is piepklein, dingen eromheen als een USB converter chip of een spanningsregelaar maken het geheel wat groter.

Een Arduino programmeer je met een computer, meestal met een laptop.
Eerst schrijf je een programma, daarna stuur je het naar de Arduino via een USB kabel.
Als het allemaal goed werkt kun je daarna de kabel weghalen en werkt jouw Arduino project op zichzelf (zoals de bedoeling is van een microprocessor) (Na het aansluiten van externe voeding!)
Er bestaat ook een Arduino Webeditor. Zie https://create.arduino.cc/
Maar deze cursus gaat uit van een vast geïnstalleerde Arduino programmeeromgeving.
Download de "installer" van https://www.arduino.cc/en/Main/Software (versie 1.8.5)
Heel vaak kun je de Arduino programma code (van iemand anders, van internet, van voorbeelden) gewoon knippen en in je eigen Arduino programma plakken.
Je hoeft het nooit allemaal over te typen! (Voor 'Knippen' en voor 'Plakken' zijn handige sneltoetsen namelijk Control-C en Control-V)

▪ Arduino en externe voeding

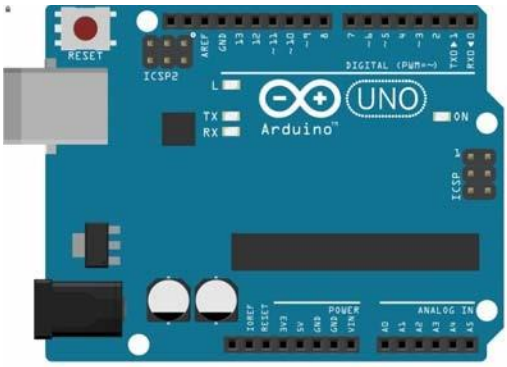
Na het uploaden is het programma in de Arduino geladen. Er is geen enkele manier om dat programma er weer "uit te krijgen". Het staat in de Arduino in 'gecompileerde' vorm opgeslagen.
De enige plek waar de oorspronkelijke bestanden staan is op de laptop (of "in the cloud" bij gebruik van de Arduino Webeditor).
Maar de USB verbinding met de laptop is niet nodig om een programma te 'draaien'.
Na het aansluiten van een externe voedingsspanning zal het laatst geladen programma automatisch starten.
Met de reset-button op de Arduino kan het desnoods opnieuw gestart worden.

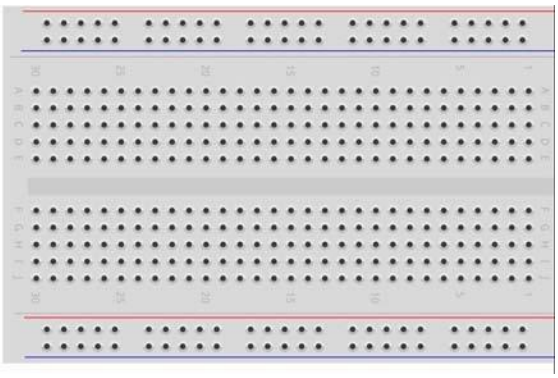
Een externe 9V batterij is een prima externe voeding, zolang er niet al te veel stroom wordt getrokken. Bij gebruik van motoren of Neopixel led's zal al snel een netadaptor nodig zijn.
De ingangsspanning moet tussen de 6 en 20 Volt liggen. Batterijvoeding vanuit 4x1.5V = 6 Volt is erg krap voor een Arduino Uno. Er is dan grote kans dat -zodra er een beetje stroom wordt gevraagd- de voedingsspanning op de Arduino onder de 5 Volt daalt en dan is het onzeker wat er gaat gebeuren. De Arduino kan ineens in reset gaan. Gebruik dus bij voorkeur tenminste 7V als voedingsspanning.
Arduino heeft een ingebouwd voedingscircuit. Ongeacht de ingangsspanning (mits boven de 6 Volt) is er altijd een stabiele 5 Volt beschikbaar als uitgang. De meeste shield werken op 5 Volt en niet op hogere spanningen! Gebruik daarom steeds de 5Volt van de Arduino op het breadbord.

▪

Het Breadboard

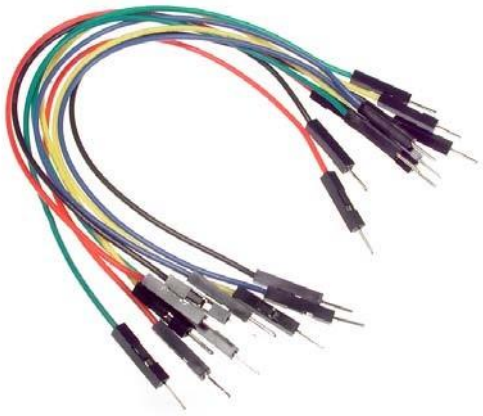
Om onderdelen met een Arduino te verbinden gebruiken we een *breadbord*.





Een breadboard is een soort gaatjesbordje waar je onderdelen kunt insteken. De bovenste twee *Rijen* en de onderste twee *rijen* zijn in het bordje vormen een horizontale verbinding.
De verticale rijen daar tussen zijn met elkaar doorverbonden. In het midden is een scheiding. Dus de bovenste helft en de onderste helft zijn niet met elkaar verbonden.

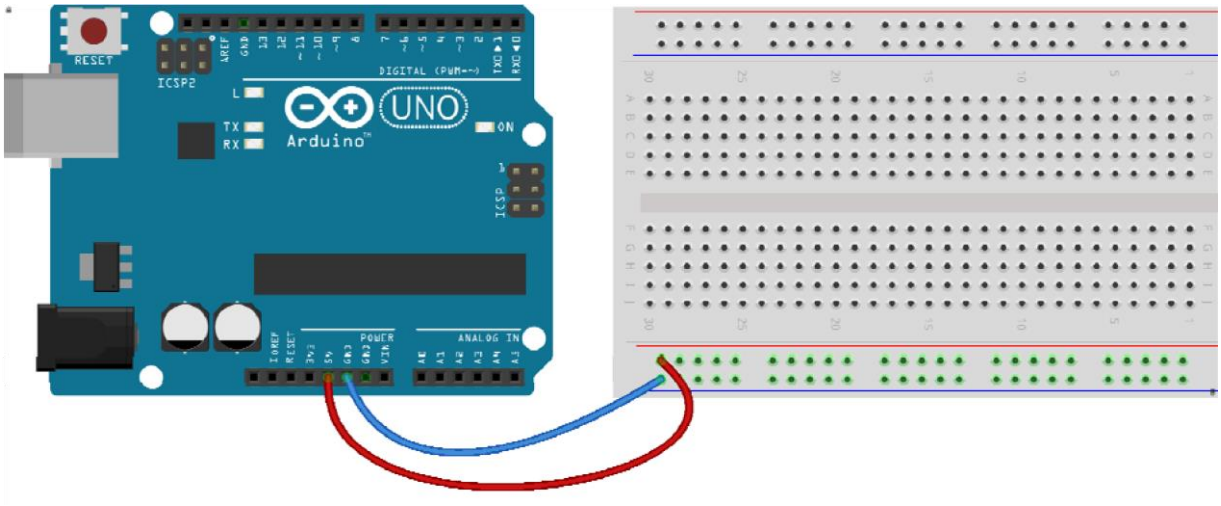
Verbindingen (tussen de Arduino en het breadbord) maken we met ‘jumpers’. Dat zijn gekleurde kabeltjes met verschillende kleuren. Er zijn “*Male-Male*” kabeltjes, “*Male-Female*” en “*Female-Female*”
Zulke verbindingen zijn vaak een bron van problemen.



Het Breadboard krijgt spanning via de Arduino. (En die krijgt weer zijn voedingsspanning via de computer of via de voedingsaansluiting (7V minimaal, 12V maximaal)

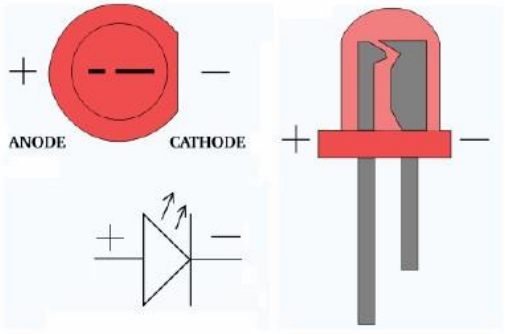
We verbinden de Gnd (*Ground)* en de 5V aansluiting van de Arduino met de rode en blauwe ‘*lijnen’* op het breadboard. Gebruik **ALTIJD** rood voor de + draad en blauw (of zwart) voor de – draad.

▪



LED's.

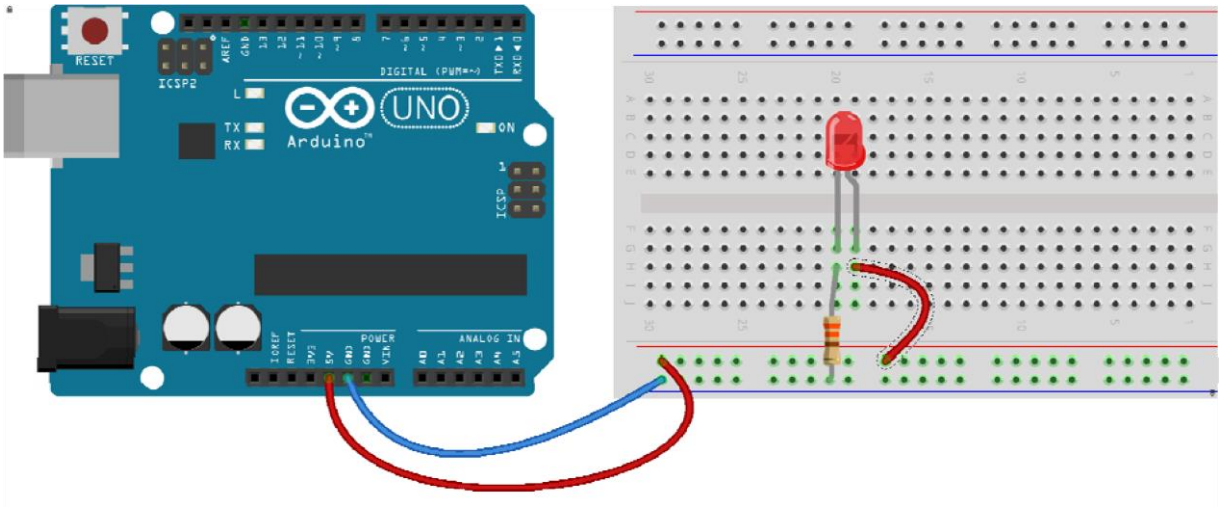
Een LED is een Light Emitting Diode, een licht uitstralende diode. Dat werkt heel anders dan een gloeilamp. Een LED heeft een + kant en een – kant. Het lange pootje is de +, het korte pootje de -
De min is ook de platte kant van de Led.



Led's kunnen doorbranden. Daarom zetten we altijd *in serie* met de Led een weerstand. Dat beperkt de stroom. Het maakt niet uit of die weerstand aan de + of aan de – kant zit.
We gebruiken bij Led's altijd een weerstand van 330 Ohm.
De waarde van een weerstand wordt met kleuren aangegeven.
Als je een weerstand zó houdt dat de goud of zilverkleurige band rechts zit dan zie je op een weerstand van 330 Ohm de kleuren Oranje-Oranje-Bruin.

Probeer eens de volgende schakeling met een rode Led en een 330 Ohm weerstand.

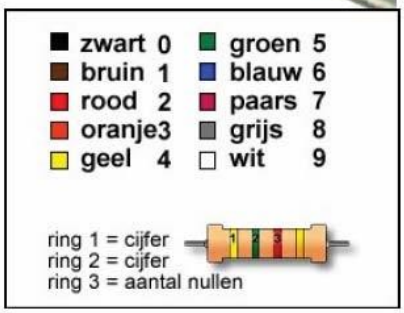
▪



Als je nu de Arduino met de laptop verbindt via de USB kabel zal de Led gaan branden.
(De Arduino doet hier nog helemaal niets behalve dat hij de spanning van de computer doorgeeft aan het breadboard.)

▪ Weerstanden:

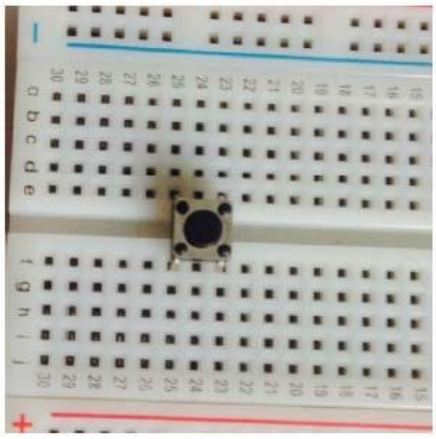
De grootte van een weerstand wordt uigedrukt in Ohm. Met de gekleurde ringen wordt de grootte van de weerstand aangegeven. Daarvoor geldt een kleurentabel.
De weerstand in het plaatje hiernaast is 1000 Ohm (1kΩ) Want: **1** omdat de eerste ring bruin is, **0** omdat de 2e ring zwart is gevolgd door **00** omdat de derde ring rood is. De vierde ring geeft de tolerantie aan. Goudkleurig staat voor 5% tolerantie.

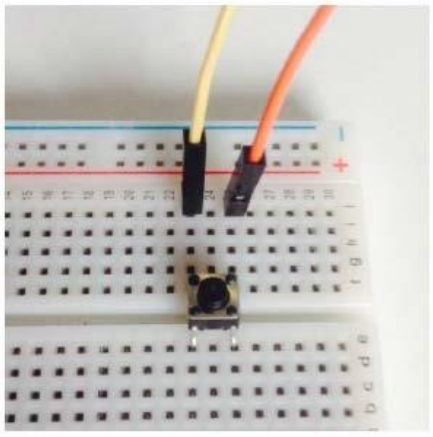


Schakelaars:

Bij de kleine en de grote drukschakelaars zijn de aansluitingen aan de linkerkant met elkaar doorverbonden. Ook de twee aansluitingen aan de rechterkant vormen een verbinding.

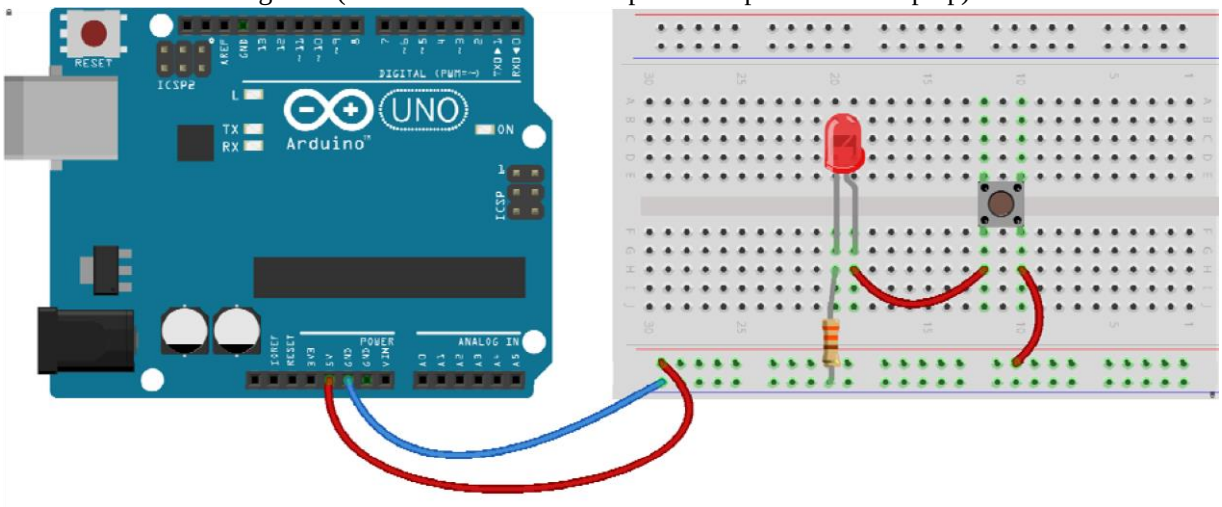
▪





Tussen de linker en de rechterkant wordt een verbinding gemaakt als je op de schakelaar drukt. Het is een *Push-button* .

Probeer deze schakeling eens (en sluit de Arduino aan op de USB poort van de laptop)



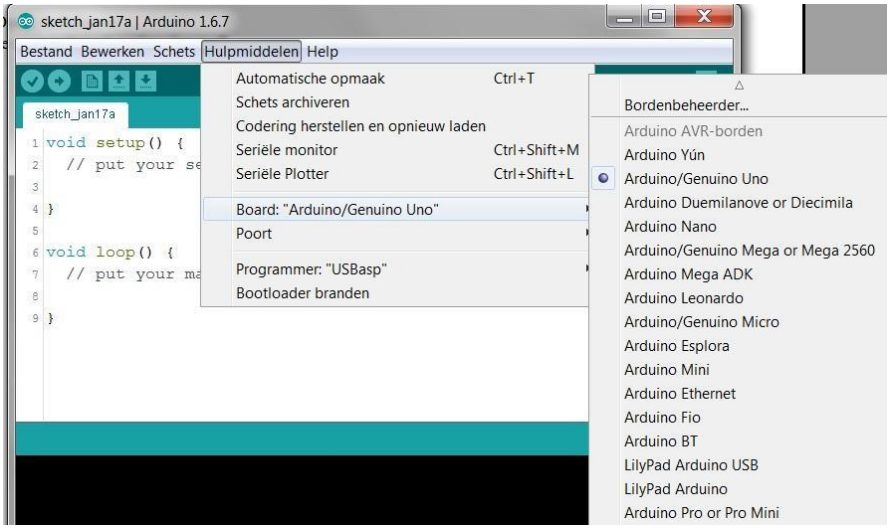
Variaties

Voeg nog een groene of blauwe Led, weerstanden en extra schakelaars toe.

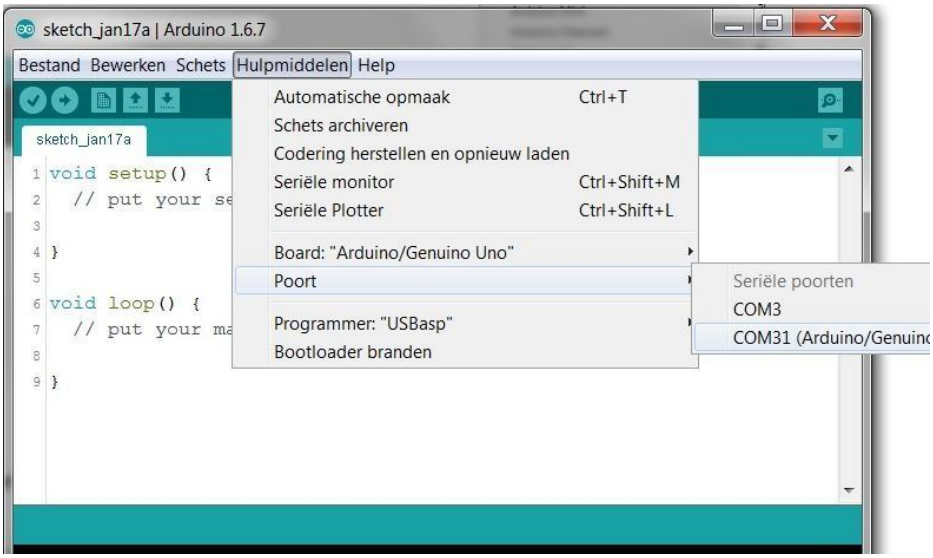
Arduino in de praktijk

Sluit je Arduino aan op een USB poort. Arduino en jouw PC gaan met elkaar communiceren via een USB kabel. Open Arduino.
Kies voor *Hulpmiddelen* en kies als *Board* voor een Arduino/Genuino Uno

▪



Kies voor *Hulpmiddelen* en controleer bij *Poort* of de laptop de Arduino 'ziet'. Als het goed is herkent de laptop vanzelf dat er een bepaalde Arduino is aangesloten.

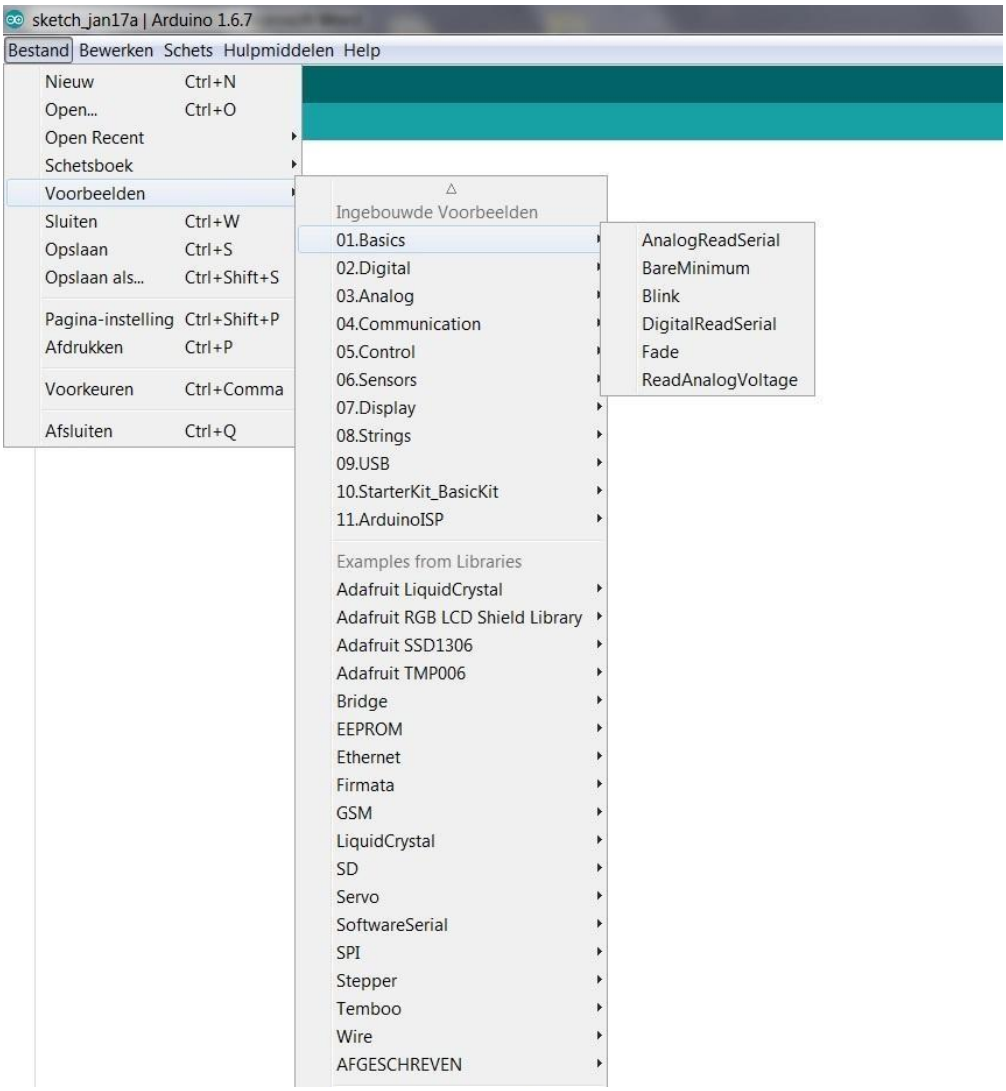


Het openen van voorbeelden:

In de Arduino programmeeromgeving zijn veel voorbeelden te vinden van basis manieren om iets te doen. Uit zulke voorbeelden zullen we stukjes naar ons eigen programma's brengen door te 'knippen' en te 'plakken'. Kies voor *Bestand* en *Voorbeelden* en je krijgt de keuze uit heel veel voorbeelden.

▪

Een handige groep programma's is *01. Basic* .
Kies (en open) het programma *Blink.*



Op je scherm zie je nu het voorbeeldprogramma *Blink* Een paar opmerkingen:

- Alles tussen /* en */ is commentaar en dat wordt door Arduino verder helemaal overgeslagen! Het is heel erg handig om in programma's veel commentaar te schrijven. Want al snel weet je anders niet meer wat er gebeurd.

- Elke *regel* die met // begint is voor Arduino ook een commentaar regel.

(Bijna) elke *programmaregel* wordt afgesloten met een ;
Een snel gemaakte fout is dat ergens een ; is vergeten. Dat levert vreemde foutmeldingen op!

- Elk Arduino programma heeft twee belangrijke blokken namelijk het blok setup en het blok loop. Zo'n blok begint met een { en eindigt ook weer met een }.
(Als je die accolades niet goed hebt staan krijg je hele vreemde foutmeldingen!) In het blok setup staan dingen die je éénmaal wilt vastleggen.
Bijvoorbeeld dat je een Led hebt vastgemaakt aan pin 13.
En b.v. dat pin 13 gebruikt gaat worden om iets naar buiten te sturen.
De opdracht daarvoor is pinMode(13, OUTPUT);

Het blok loop wordt door de Arduino voortdurend 'doorlopen'. Alles wat daar staat wordt voortdurend, zonder ooit te stoppen herhaald en uitgevoerd door de Arduino. Er staan een paar regels die veel op elkaar lijken.

digitalWrite(13, HIGH);	Hiermee wordt pin 13 'Hoog' gemaakt. Dat is hetzelfde als 'met plus verbonden'. De Led gaat aan. Let op! ELKE regel wordt in Arduino met ; afgesloten!
delay(1000);	De opdracht om 1000 milliseconden helemaal niets te doen
digitalWrite(13, LOW);	Hiermee wordt pin 13 'Laag' gemaakt. Dat is hetzelfde als 'met min verbonden'. De Led gaat uit.
delay(1000);	De opdracht om 1000 milliseconden helemaal niets te doen

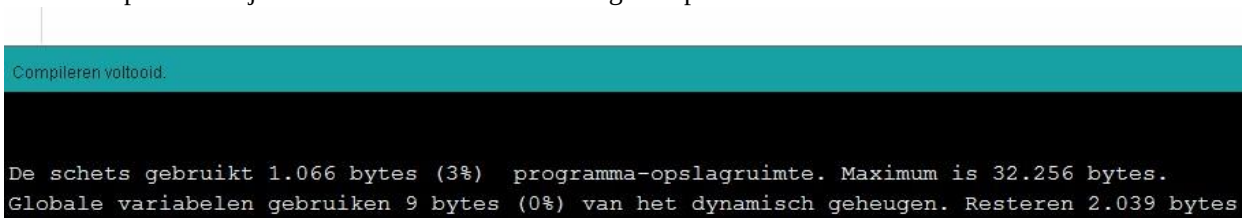
We hebben dus een programma waarmee een Led die verbonden is met pin 13 gedurende 1 sec. aan gaat en 1 sec. uit! En bij een Arduino is er altijd een Led verbonden met pin 13. Die zit óp de Arduino.

Een *Knipperled* met het voorbeeld programma *Blink*

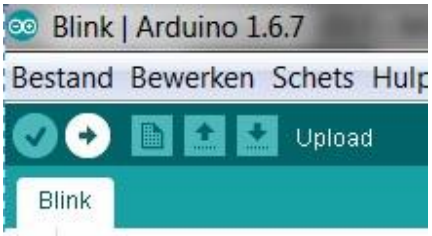
```
/*
   Zet een LED aan voor 1 seconde en daarna uit voor 1 seconde. En herhaal dit. */

// De Setup functie wordt één keer uitgevoerd als je het board aanzet.

void setup() {
  // Maak van pin 13 een Output pin.
  pinMode(13, OUTPUT);
}

// De Loop functie wordt eindeloos lang doorlopen.
void loop() {   digitalWrite(13, HIGH);   // Zet de Led aan. (HIGH is hetzelfde als
verbinden met +)   delay(1000);                  // Wacht 1 seconde    digitalWrite(13,
LOW);    // Zet de Led uit. (LOW is hetzelfde als verbinden met -)   delay(1000);
// Wacht 1 seconde  }
```

We gaan programma Blink "*Uploaden*" naar de Arduino.
Maar eerst moet Arduino controleren of het programma in correcte programmeertaal is geschreven. Die controle heet *Compileren.*
Onder de menubalk (Bestand Bewerken Schets Hulpmiddelen enz. staan twee 'knopjes'. De meest linker is voor "Compileren".

Na het compileren zie je onderin het scherm de melding Compileren voltooid.



Als er geen foutmeldingen worden gegeven kunnen we nu gaan *Uploaden* (het tweede knopje)



Tijdens het *Uploaden* wordt het programma nogmaals gecompileerd en daarna naar de Arduino gestuurd. Als alles goed gaat verschijnt onderin je scherm de melding Uploaden voltooid.

Het Ledje op de Arduino knippert 1x per seconde.

Variaties:
Verander de delay waarden in de loop. Maak er bijvoorbeeld een flitslicht van.

```
void loop() {   digitalWrite(13, HIGH);   // Zet de Led aan. (HIGH is hetzelfde als
verbinden met +)   delay(50);                    // Wacht 50 mseconde    digitalWrite(13,
LOW);    // Zet de Led uit. (LOW is hetzelfde als verbinden met -)   delay(1000);
// Wacht 1 seconde  }
```

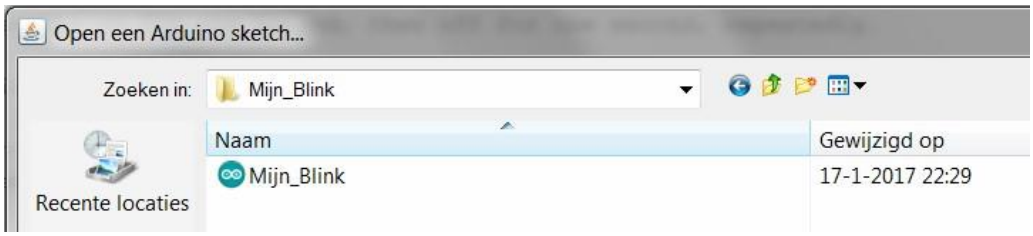
Opslaan van Arduino bestanden

Als we het flitslicht, (de aangepaste '*Blink'*) willen opslaan dan lukt dat niet direct.
Want Blink was een voorbeeld, dat kunnen we niet veranderen.
Kies voor:
Bestand, Opslaan als, blader naar je USB stick en type als naam in: *Mijn Blink* En
nu gebeuren er verschillende dingen.

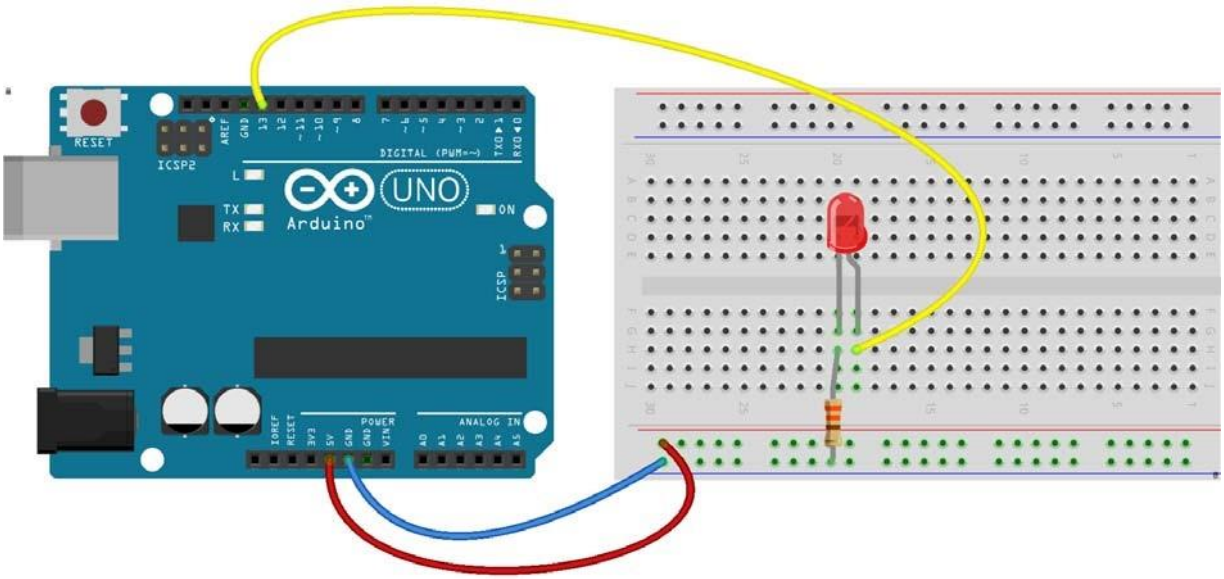
- Arduino accepteert geen naam met een spatie en maakt er *Mijn_Blink* van.
- Arduino maakt een *map* aan met de naam *Mijn_Blink.*
- Arduino slaat in die map het bestand op als *Mijn_Blink.ino* Een programma in Arduino heet een sketch.

Openen van Arduino bestanden

Kies voor:
Bestand, Openen, blader naar je USB stick en daar zie je een map *Mijn_Blink.* Klik er op en je ziet in die map het programma *Mijn_Blink* staan. Door er op te klikken open je het bestand.
(Een handige manier om één van je laatste programma's te openen is Open Recent)



Nu gaan we het programma zo aanpassen dat de Led op het breadboard gaat knipperen. Verbind de + kant van de Led met pin 13.



Er is niets veranderd aan het programma. Pin 13 wordt "*even hoog*", wordt daarna weer *"even laag*". De Led knippert mee met de Led op de Arduino.

Libraries

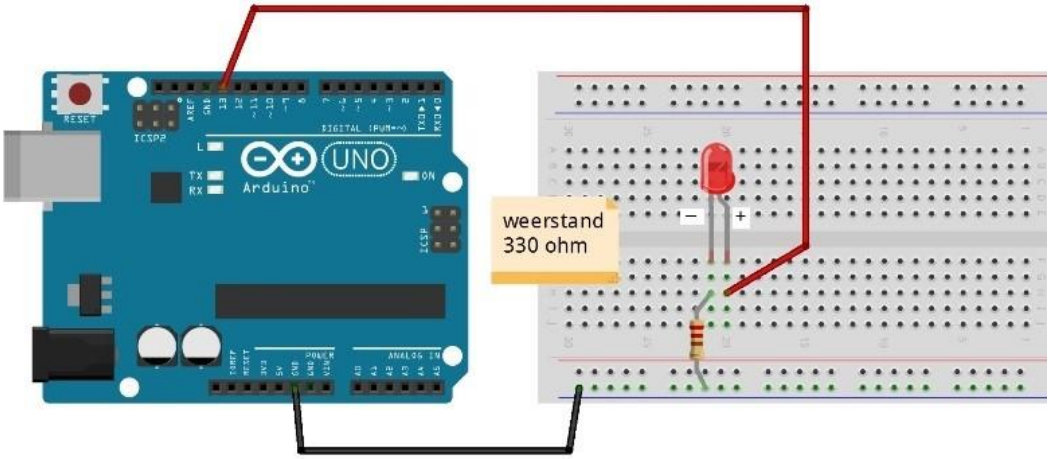
In veel Arduino programma's wordt gebruik gemaakt van een bibliotheek. Een *library*.
Bij deze cursus hoort een USB met bestanden en alle benodigde libraries.
Alle bestanden kunnen ook online worden gevonden op https://tinyurl.com/y88ugf2o De inhoud van die USB stick en de map op internet is:

Libraries moeten geplaatst worden in C:\Program Files\Arduino\libraries óf, -beter eigenlijk- in C:\users\usernaam\Documents\Arduino
Na het wijzigen of plaatsen van een library moet Arduino opnieuw gestart worden! Zie ook https://www.arduino.cc/en/Guide/Libraries

Projecten

Project 1: Een knipperled.

Aansluitingen:



Variaties:

Verander de ‘Aan-tijd’, de ‘Uit-tijd’
Verander in de schakeling de aansluiting op pin 13 in pin 12.
Verander ook in het programma pin 13 in pin 12.

Je merkt dat je daarvoor op verschillende plaatsen die ‘*13’* in een ‘*12’* moet veranderen.
Dat kan handiger! We gaan werken met een *variabele* die we de naam geven *LedPin.*
Nog vóór het blok setup voeren we de naam, het type en de waarde van deze *variabele* in.

Het programma:

```
int LedPin=13;

void setup() {
  pinMode(LedPin, OUTPUT);
}

void loop() {
  digitalWrite(LedPin, HIGH);
  delay(50);
  digitalWrite(LedPin, LOW);     delay(1000);
}
```

De uitleg:

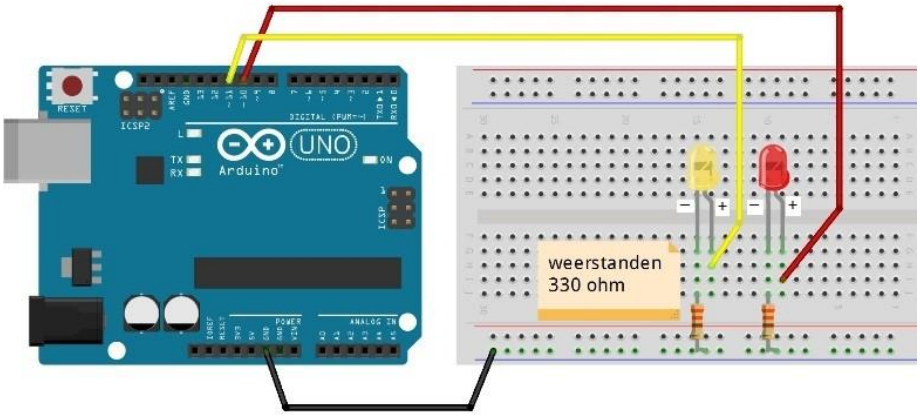
De eerste regel is int LedPin=13; *int* betekent dat we voor LedPin een
getal geven dat een geheel getal is.
Arduino kent nog meer soorten getallen.
Een *byte* is een geheel getal tussen de 0 en de 255, een *boolean* is een 0 of een 1. (Of LOW of HIGH) En je kunt ook wel met kommagetallen werken. Maar die vragen veel van de schaarse geheugenruimte.

Doordat we van LedPin een *variable* hebben gemaakt kunnen we nu snel ons programma en onze schakeling wijzigen. Je hoeft nu alleen die Led op pin 12 aan te sluiten, LedPin=12 te maken, en je programma natuurlijk weer te uploaden. Nu knippert de Led op het Arduino bord niet meer mee. (Want die zit nog steeds vast aan 13.)

Nog twee kleine opmerkingen:
In deze schakeling is die Led aan b.v. pin 10 vastgemaakt. Dat verandert nooit meer in het programma! Dan is het beter om te schrijven: `const int LedPin=10;`
Een andere manier is: `#define LedPin 10` Let op! Deze regel is zónder afsluitende `;` Nu weet Arduino dat LedPin één keer een waarde krijgt en daarna niet meer verandert. Dat scheelt geheugen om de waarde van LedPin te bewaren.

Het tweede probleem is moeilijker. Tijdens een *Delay* doet Arduino helemaal niets. Luistert nergens naar, reageert nergens op, maar wacht tot die delaytijd voorbij is. Daar is een oplossing voor maar die moet nog even wachten.

Project 2: Meerdere knipper LED's!

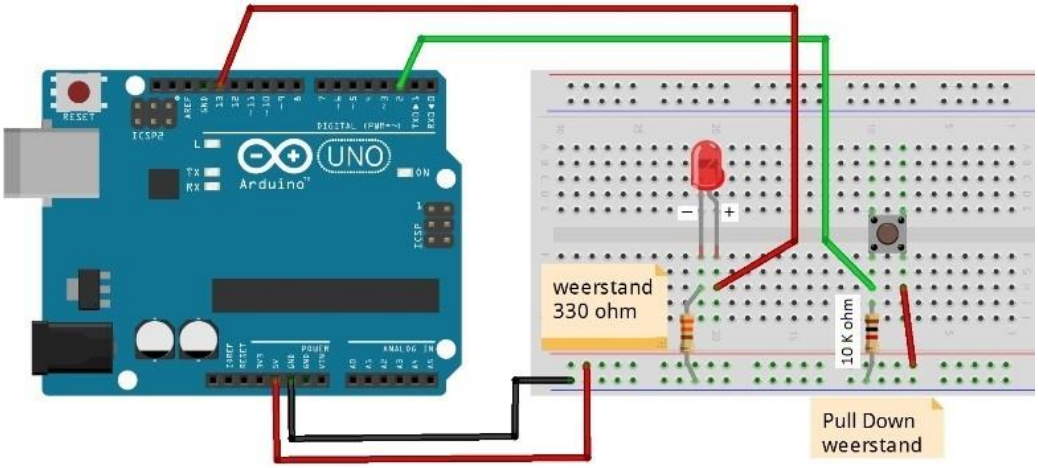


Variaties:

- Laat de twee Leds om en om knipperen.
- Plaats Monteer op het breadboard een derde, groene Led bij.
  Sluit de groene Led aan op b.v. pin 12.
  Laat deze derde Led ook mee knipperen.
- Maak van je opstelling een stoplicht (rood, oranje, groen)

-

Project 3: LED met schakelaar

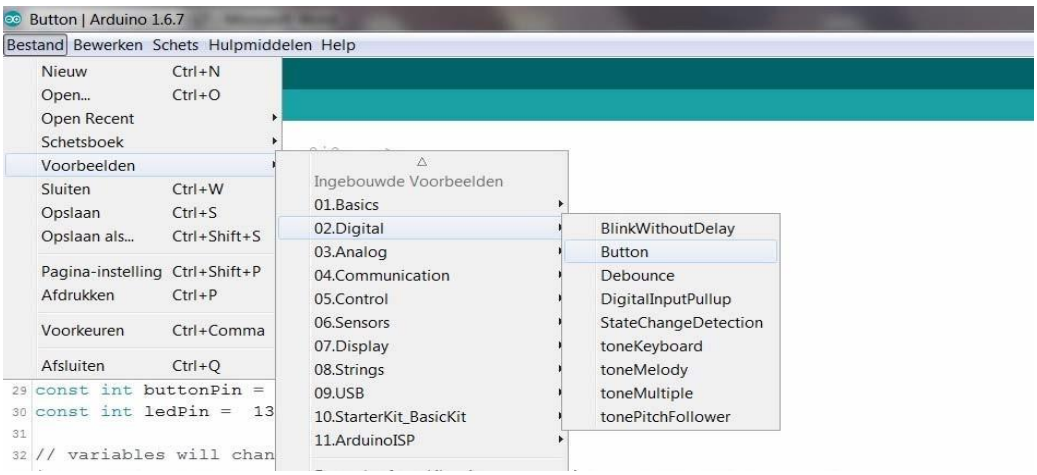


Verbind één kant van de schakelaar met een weerstand van 10.000 Ohm aan de – lijn.
10.000 Ohm wordt korter geschreven als 10k (want k is kilo).
De kleurencode is Bruin-Zwart-Oranje. (Want Bruin is 1, Zwart is 0 en Oranje is 3)

Resultaat:

Als de schakelaar niet is ingedrukt is die verbindingsdraad via de weerstand verbonden met de – De weerstand 'trekt' die draad naar beneden. Dit heet een Pull-Down weerstand.
Als je de schakelaar indrukt wordt verbindingsdraad (dus ook pin2) verbonden met de +.
Dus niet ingedrukt: pin2 is *Laag*
Wel ingedrukt: pin2 is *Hoog*

We gaan nu zorgen dat Arduino het verschil ziet.
Kies voor Bestand, Voorbeelden, 02.Digital, Button



```
const int buttonPin = 2;     // the number of the pushbutton pin const
int ledPin =  13;      // the number of the LED pin

int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH) {
digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

Sommige regels kwamen al eerder voor.

- De schakelaar is vastgemaakt aan pin 2
- Er wordt weer gebruik gemaakt van de ingebouwde Led op pin 13. const int buttonPin = 2; // the number of the pushbutton pin const int ledPin = 13; // the number of the LED pin
- Er is een variabele en een naam ingevoerd voor de Toestand van de schakelaar.
  Ingedrukt is 1, niet ingedrukt is 0.
  Omdat dit in het programma kan veranderen kan hier niet const voor staan.
  int buttonState = 0; // variable for reading the pushbutton status - In de setup wordt opnieuw ledPin als Output gedefinieerd.
  Nieuw is dat de buttonPin als Input wordt gedefinieerd.
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
- In de loop wordt continue de toestand van de schakelaar gelezen met de opdracht digitalRead.
  buttonState = digitalRead(buttonPin);

En nu laten we de Arduino een beslissing nemen.
Als buttonState = LOW, (0 kan ook) dan is de schakelaar niet ingedrukt en moet de Led uitgezet.
Als buttonState = HIGH, (1 kan ook) dan is de schakelaar wel ingedrukt en moet de Led aangezet.
Die beslissing staat in een if(voorwaarde) { doe dan dit } else {doe iets anders} Let op het verschil tussen de volgende 2 regels!
buttonState = HIGH betekent dat buttonState HIGH wórdt. buttonState == HIGH betekent de vraag of buttonState HIGH ís.

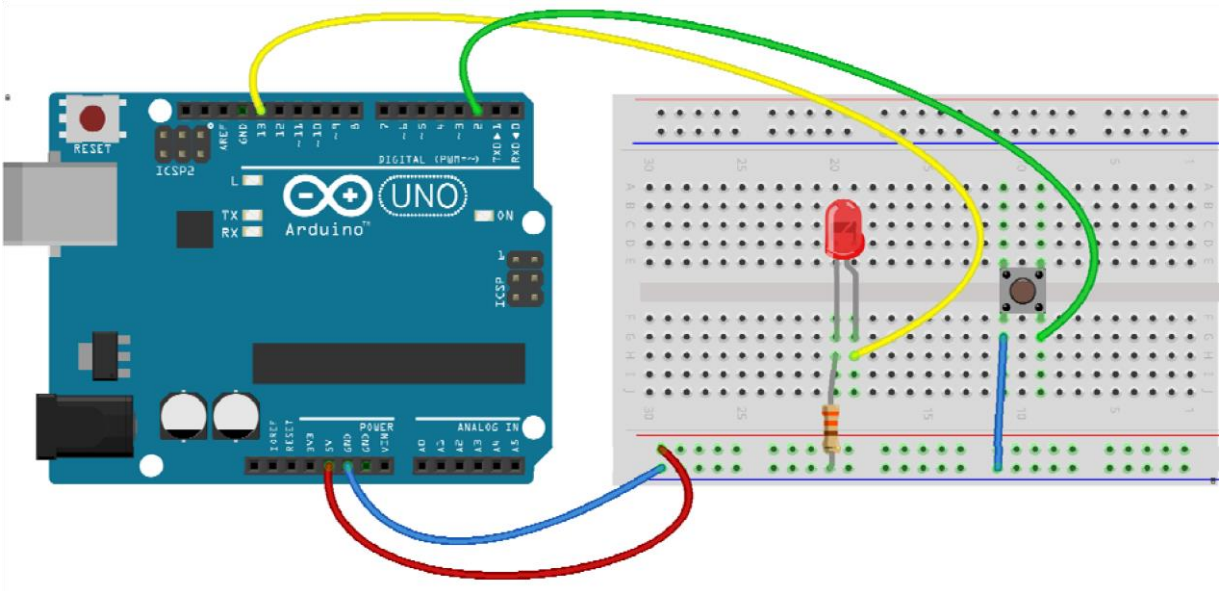
```
  if (buttonState == HIGH) {
digitalWrite(ledPin, HIGH);    // turn LED on:
  }    else {                digitalWrite(ledPin, LOW);    // turn
LED off:   }
```

Na het uploaden van programma *Button* zal de Led aangaan als op de schakelaar wordt gedrukt.

Pull_Down of Pull_Up

Het gebruik van zo'n *Pull-Down* weerstand is onhandig. Het breadboard wordt er onoverzichtelijk door. Het kan ook anders.

In plaats van een Pull-Down weerstand kunnen we netzogoed een Pull-Up weerstand gebruiken.
Dan '*trekken'* we de schakelaar naar de + en bij indrukken wordt het een – Dat lijkt niets uit te maken maar… die Pull_Up weerstand zit al in de Arduino.



Verander hiervoor in het programma de regel pinMode(buttonPin, INPUT);

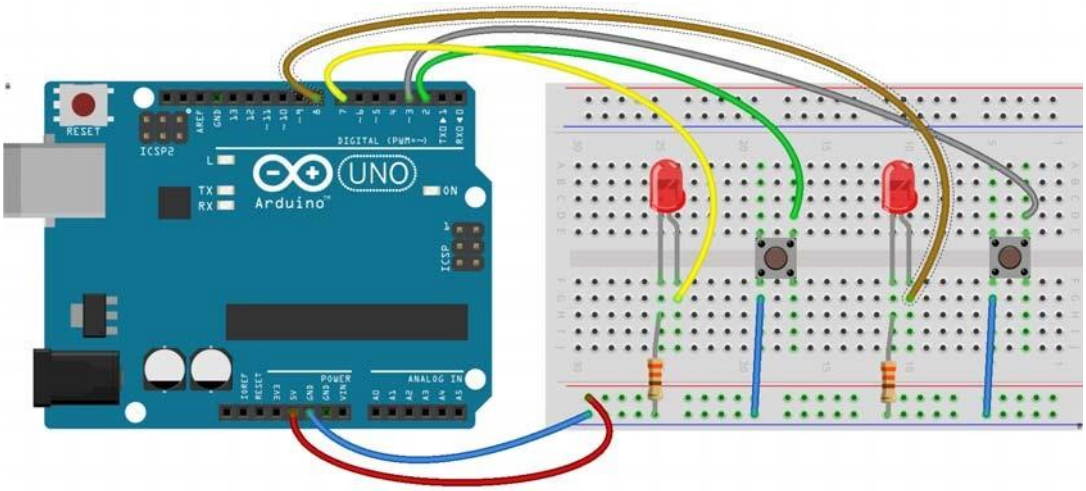
in
pinMode(buttonPin, INPUT_PULLUP);

Dit "activeert" de interne Pull_Up weerstand. De schakeling wordt eenvoudiger.
Maar …..
De Led is nu AAN als je de schakelaar niet indrukt en gaat juist UIT als je 'm wel indrukt!
Dat komt omdat het indrukken van de schakelaar nu de buttonPin naar de – trekt in plaats van naar de +.
Dat is makkelijk op te lossen in je programma. Verander HIGH in LOW.

```
 if (buttonState == LOW) {
digitalWrite(ledPin, HIGH);
 } else {
        digitalWrite(ledPin, LOW);
}
```

Project 4: Twee LED's met twee schakelaars

Als de ene knop ingedrukt wordt gaat de ene Led aan, enz.



Het programma wordt dubbel zo groot om twee Led's en twee schakelaars goed te krijgen.
Het is een goed plan om "betekenisvolle" namen te gebruiken.
Denk er aan dat Arduino hoofdletter gevoelig is! RedledPin is niet hetzelfde als Redledpin

```
const int LeftbuttonPin = 2;      const
int RightbuttonPin = 3;    const int
RedledPin =  7;
const int GreenledPin =  8;

int LeftbuttonState = 0;
int RightbuttonState = 0;

void setup() {
  pinMode(RedledPin,    OUTPUT);   pinMode(GreenledPin,
OUTPUT);     pinMode(LeftbuttonPin,   INPUT_PULLUP);
pinMode(RightbuttonPin, INPUT_PULLUP);
}

void loop() {
  LeftbuttonState   = digitalRead(LeftbuttonPin);
  RightbuttonState = digitalRead(RightbuttonPin);

  if (LeftbuttonState == LOW) {
digitalWrite(RedledPin, HIGH);
  } else {
    digitalWrite(RedledPin, LOW);
  }
    if (RightbuttonState == LOW) {
digitalWrite(GreenledPin, HIGH);
  } else {
    digitalWrite(GreenledPin, LOW);
  }
}
```

Project 5: De FOR opdracht.

Soms willen we een bepaalde gedeelte in het programma een aantal keren laten uitvoeren.
Een Ledje moet bijvoorbeeld 3 keer knipperen.
Dan moet het programma bijhouden hoe vaak het al is uitgevoerd. We hebben dus een tellertje nodig.
Tellertjes komen heel veel voor in computerprogramma's.
Het volgende programma laat een led precies 3 keer knipperen. (Je kunt beter een andere ledpin gebruiken dan 13… Want tijdens het uploaden knippert de ingebouwde led op pin 13 ook..)

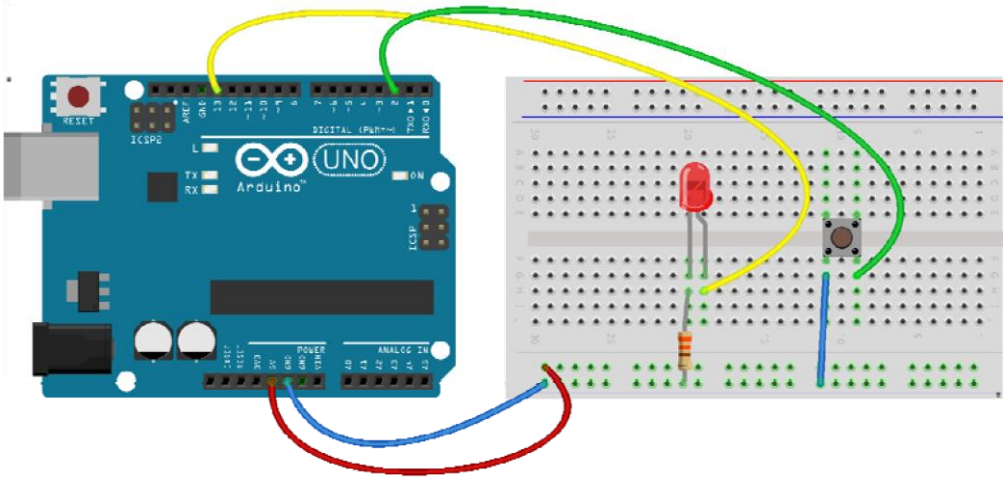
```
int ledpin = 13;

void setup() {
  pinMode(ledpin, OUTPUT);

  for (int i = 1; i <= 3; i++) {
    digitalWrite(ledpin, HIGH);   // turn the LED on
delay(100);                 // wait
    digitalWrite(ledpin, LOW);    // turn the LED off
delay(1000);                 // wait
  }
}

void loop() {
}
```

De regel waar het om gaat is:

```
  for (int i = 1; i <= 3; i++) {
 }
```

Hier staat dat er een variable *i* is waarmee we gaan tellen.
Die variable is nog niet eerder gebruikt dus we vertellen Arduino dat het een *int* is, een geheel getal. *i* begint met waarde 1. Alles tussen de { } wordt een keer uitgevoerd.
Daarna wordt *i* met één opgehoogd. Dat kan in Arduino met een hele korte opdracht. *i*++ En alles tussen de accolades wordt opnieuw uitgevoerd, en opnieuw en opnieuw….
Elke keer controleert Arduino of er voldaan is aan de voorwaarde dat i nog steeds kleiner of gelijk moet zijn aan 3. i <= 3 betekent *i* □3
Zodra dat niet langer klopt wordt deze 'lus' niet langer uitgevoerd.
In dit voorbeeld staan deze regels allemaal in de setup() Als ze in de loop() zouden staan gaat het ledje 3 keer aan en uit, maar daarna wordt dat weer herhaalt want alles de loop() wordt eeuwig herhaalt!

Project 6: Een Flash led met drukknop.

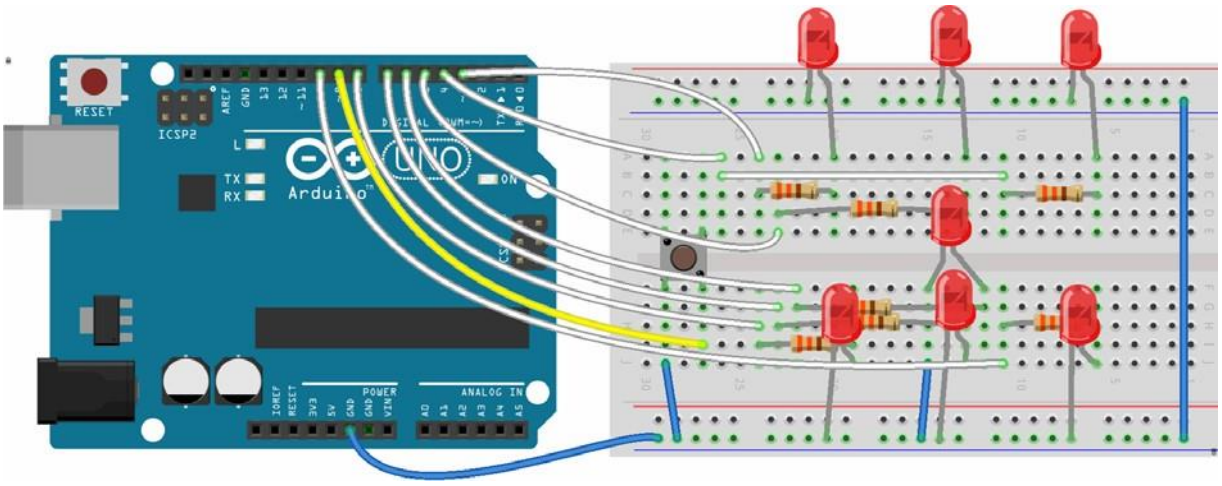


Probeer te begrijpen wat er in dit programma gebeurt.

```
int buttonPin = 2;      // the number of the pushbutton pin int
ledPin =  13;       // the number of the LED pin
int buttonState = 1;   // variable for reading the pushbutton status

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

void loop() {
  buttonState = digitalRead(buttonPin);
if (buttonState == LOW) {     for (int i
= 1; i <= 3; i++) {
     digitalWrite(ledPin, HIGH);   // turn the LED on
     delay(50);                    // wait
     digitalWrite(ledPin, LOW);   // turn the LED off
     delay(500);                   // wait
}   }
}
```

Project 7: Een electronische dobbelsteen

Een uitdagend project waar alle vorige kennis in samenkomt is het maken van een electronische dobbelsteen.
Een dobbelsteen heeft 7 ogen, er zijn dus 7 Led's nodig. Maar.. sommige Led's gaan altijd met 2 tegelijkertijd aan of uit, b.v. de diagonale Led's en de 2 middelste Led's. Dat betekent dat we eigenlijk geen 7 Led's hoeven aan te sturen maar slechts 4 om toch alle "ogen" te kunnen gooien. Maar de grote uitdaging is om dat alles op het breadbord geplaatst en verbonden te krijgen!
In dit voorbeeld is gekozen om toch alle led's apart aan te sluiten. (We hebben pinnen genoeg…)



Bij zo'n breadbord vol is het wel handig om eerst wat eenvoudige testjes uit te voeren om te zien of alles goed is aangesloten.
Het kan zelfs zonder programmeren… Neem één voor één die verbindingen met de Arduino los en tik even de 5V aan. Dan moet die Led aan gaan.

In het programma staat niets nieuws.
De schakelaar wordt ingelezen en als-ie ingedrukt is …. wordt een toeval getal gekozen!

```
   value = random(1, 7); //Kies een getal van 1 t/m 6. (7 doet niet meer mee)
if (value == 1) {
    digitalWrite (pinLed7, HIGH);
   }
   if (value == 2) {
    digitalWrite (pinLed3, HIGH);
digitalWrite (pinLed4, HIGH);
   }
   if (value == 3) {      digitalWrite
(pinLed3, HIGH);      digitalWrite
(pinLed4, HIGH);      digitalWrite
(pinLed7, HIGH);
   }
```

Maar hoe kan een 'automaat' nu ooit een willekeurig toeval getal kiezen? Er ligt hoe dan ook een of ander principe onder waardoor je steeds hetzelfde toeval getal zou moeten krijgen.
Om toch 'vrijwel echt' toeval te krijgen wordt de toevalsgenerator eerst 'gezaaid' met een echt willekeurig getal, namelijk de ruis op de analoge poort A0. (Komt pas in Project 11 aan de orde maar we gebruiken het nu alvast wel…In de setup staat de opdracht randomSeed(analogRead(0)); en deze 'zaait de toevalsgenerator. Resultaat van dit programma. Na een druk op de knop 'gooit' de dobbelsteen een getal.
De essentiële regels van het programma:

```
int pinLed1 = 3; int
pinLed2 = 5; int
pinLed3 = 4; int
pinLed4 = 8; int
pinLed5 = 7; int
pinLed6 = 9; int
pinLed7 = 6;
int buttonPin = 10;

void setup ()
{   pinMode (pinLed1, OUTPUT);   enz.
enz.   pinMode (buttonPin,
INPUT_PULLUP);
  randomSeed(analogRead(0));  //Om het toevalsproces van de functie random te starten }

void loop()
{   int value;   if (digitalRead(buttonPin) == LOW) { // Ja! De schakelaar
is ingedrukt.     value = random(1, 7); //Kies een getal van 1 t/m 6     if
(value == 1)    digitalWrite (pinLed7, HIGH);     if (value == 2) {
digitalWrite (pinLed3, HIGH);                    digitalWrite (pinLed4,
HIGH);
    }
   if (value == 3) {   digitalWrite (pinLed3, HIGH);
digitalWrite (pinLed4, HIGH);                    digitalWrite
(pinLed7, HIGH);
    }
   if (value == 4) {   digitalWrite (pinLed1, HIGH);
digitalWrite (pinLed3, HIGH);                    digitalWrite
(pinLed4, HIGH);                    digitalWrite (pinLed6,
HIGH);
    }
   if (value == 5) {   digitalWrite (pinLed1, HIGH);
digitalWrite (pinLed3, HIGH);                    digitalWrite
(pinLed4, HIGH);                    digitalWrite (pinLed6,
HIGH);                    digitalWrite (pinLed7, HIGH);
    }
   if (value == 6) {   digitalWrite (pinLed1, HIGH);
enz. enz.
    }
   delay(1000);    // Zet de Led's na 1 seconde weer uit
   digitalWrite     (pinLed1,     LOW);
enz. enz.
  }
}
```

Op deze dobbelsteen zijn nog heel wat variaties te bedenken.
Een echte dobbelsteen rolt een aantal keren, steeds langzamer en komt dan pas tot stilstand.
Dat kan ingebouwd worden.
Het aantal keren zou ook weer ‘random’ kunnen zijn, b.v. minstens 3x, maximaal 7 keer.
En het ‘beepertje’ uit project 16 zou hier ook gebruikt kunnen worden. Als de dobbelsteen is uitgerold klinkt b.v. een “TaDa, TaDáá”.
Een andere variatie is het vervangen van de schakelaar door een *tilt-schakelaar*.

Project 8: Functies

Programma's moeten zo simpel mogelijk zijn. Een manier om dat te doen is om een stel regels die bij elkaar horen en samen "iets doen" ook als groep bij elkaar te stoppen.
Het volgende programma doet precies hetzelfde als het programma hierboven. (Het eerste gedeelte tot en met *setup()* is precies hetzelfde). Maar dit programma is makkelijker te lezen. Als je op de knop drukt moet de functie *ledknipper* uitgevoerd worden.

```
int buttonPin = 2;     // the number of the pushbutton pin int
ledPin =  12;       // the number of the LED pin
int buttonState = 1;  // variable for reading the pushbutton status

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

void loop() {
  buttonState = digitalRead(buttonPin);
if (buttonState == LOW) {
    ledknipper();
  }
}

void ledknipper() {    for (int
i = 1; i <= 5; i++) {
    digitalWrite(ledPin, HIGH);   // turn the LED on
delay(25);                 // wait
    digitalWrite(ledPin, LOW);    // turn the LED off
delay(200);                  // wait
  }
}
```

Het kan nog korter. Na die *if*.... volgt maar één opdracht. Dan (alleen dan) mag je de accolades weglaten. Dat maakt alles nog overzichtelijker. En die variablele buttonState kan er ook wel uit. Maar met korter wordt er aan de andere kant ook niet altijd duidelijker op….

```
 if (digitalRead(buttonPin) == LOW) ledknipper();
```

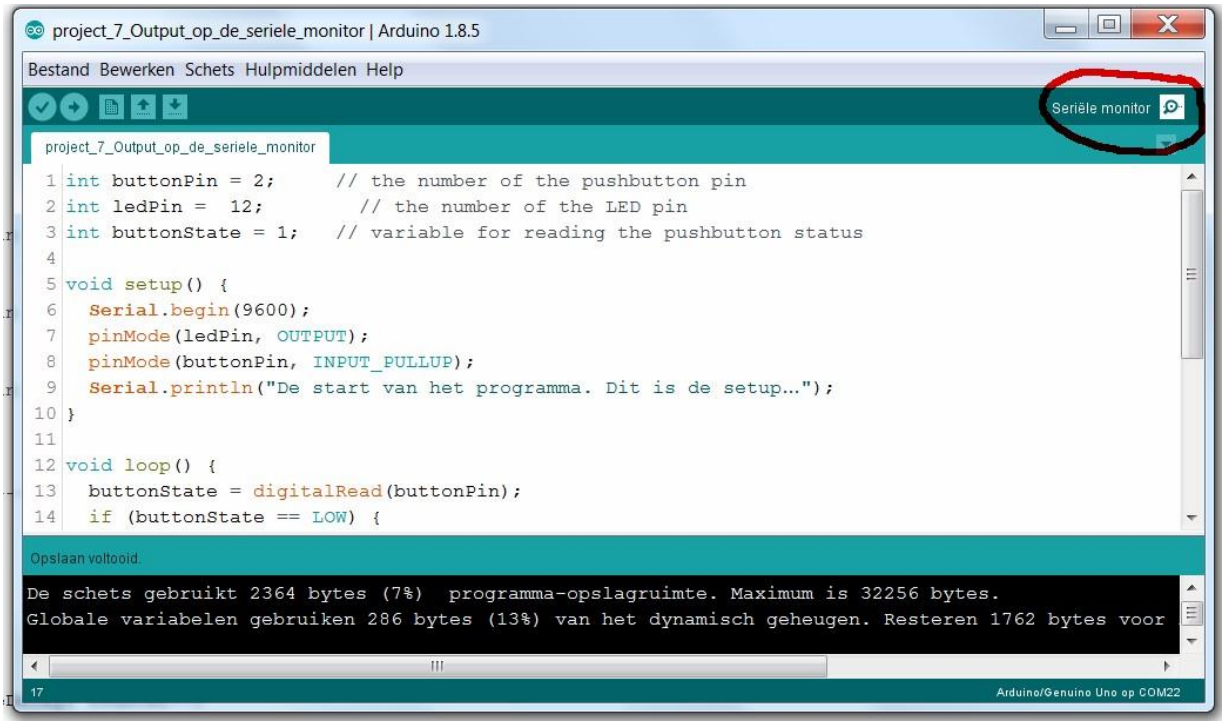
Project 9: Output op de seriële monitor.

Onze programma's worden steeds ingewikkelder. Maar het enige wat we zien is dat ene Ledje.
Doet het wat we willen of niet? En als het niet aangaat, komt dat door een fout in het programma of is het ledje of de schakelaar misschien kapot? We willen meer hulpmiddelen om te zien wat er in ons programma gebeurt!
Eén manier is het weergeven van resultaten op de laptop. Dan moet de Arduino via de USB kabel informatie terug sturen. Dat werkt zó… (Gebruik dezelfde schakeling als in het vorige project)

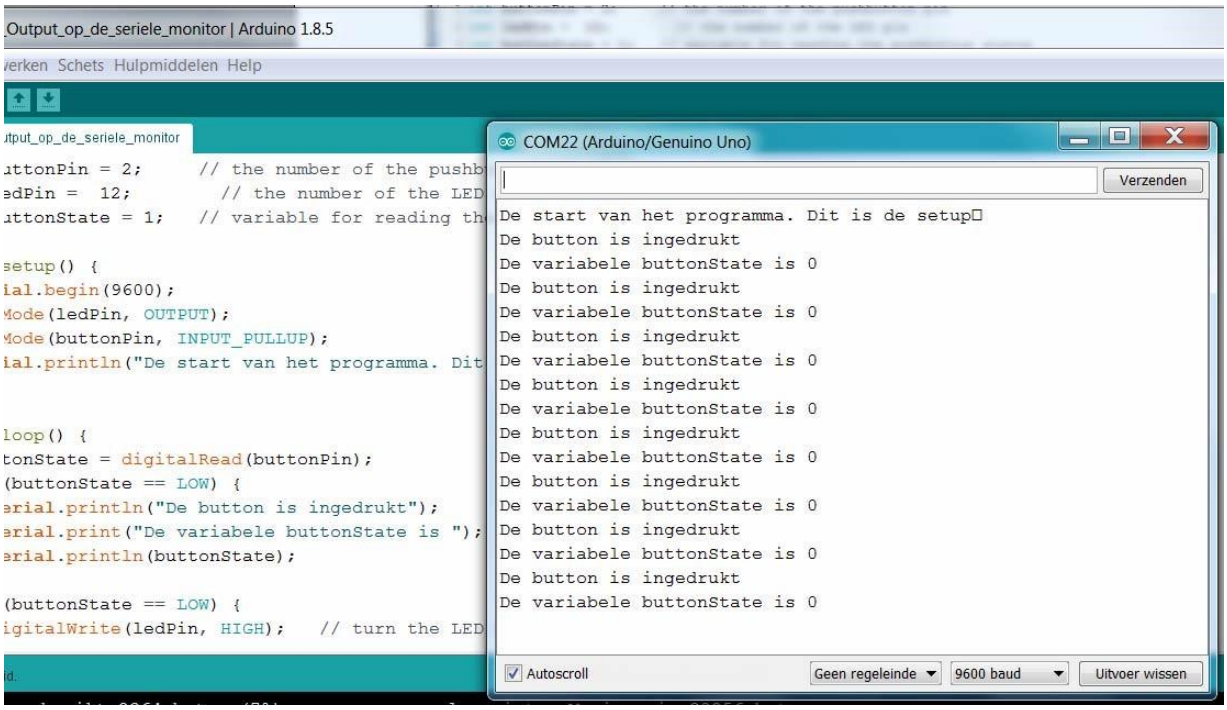
```
int buttonPin = 2;     // the number of the pushbutton pin int
ledPin =  13;       // the number of the LED pin
int buttonState = 1;   // variable for reading the pushbutton status

void setup() {
  Serial.begin(9600);
pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
  Serial.println("De start van het programma. Dit is de setup….."); }

void loop() {
  buttonState = digitalRead(buttonPin);
  if (buttonState == LOW) {
    Serial.println("De button is ingedrukt");
    Serial.print("De variabele buttonState is ");
    Serial.println(buttonState);

  }
  if (buttonState == LOW) {
    digitalWrite(ledPin, HIGH);   // turn the LED on
  }
else {
    digitalWrite(ledPin, LOW);   // turn the LED off
  }
}
```

- In de setup wordt met Serial.begin(9600);verteld op welke snelheid de Arduino en de laptop met elkaar 'communiceren'.
- Daarna sturen we informatie naar de laptop met opdrachten als Serial.println("De buttonstatus is "); en Serial.println(buttonState);
  De 2e opdracht geeft tegelijk de opdracht "ga naar een nieuwe regel", de 1e opdracht doet dat niet… Meestal niet zo handig.

Maar, hoe krijg je die Seriële 'Output' op het scherm?
Ga ná het uploaden van dit programma naar "*Seriële monitor"* rechts bovenin.

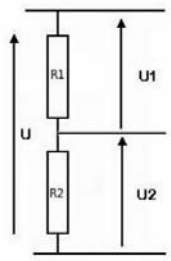


Je ziet nu de output van je programma op de seriële monitor. Je kunt wat beter zien wat er allemaal gebeurd en dat het programma de schakelaar inderdaad ‘leest”.

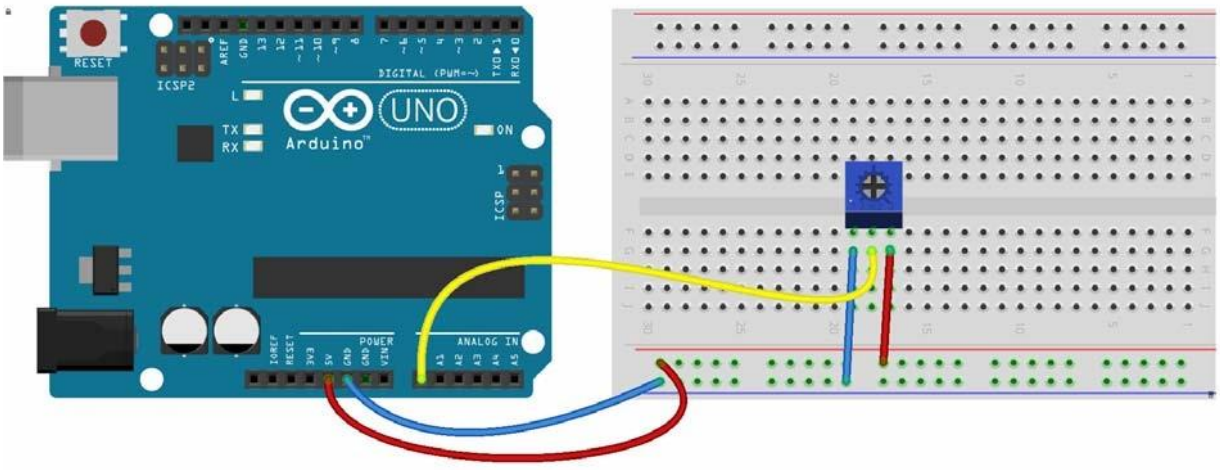


Project 10: Meten van een spanning op de analoge ingang

Arduino heeft 6 analoge ingangen (A0 t/m A5) waarmee gemeten kan worden.
Op een ingang van de Arduino mag maximaal 5 Volt aangesloten worden.
Sluit een potmeter (instelbare weerstand) van 10k aan zoals in de figuur.
We meten de spanning van de middelste aansluiting, dit heet de loper van de potmeter.
Door te draaien aan de potmeter veranderen we de weerstand links en rechts van de loper.
De totale weerstand van beide weerstanden bij elkaar opgeteld blijft gelijk. In dit geval is dit 10 K Ohm.
Een potmeter wordt een spanningsdeler genoemd.
Door aan de knop te draaien van de potmeter, varieert de spanning van 0 V tot 5 V



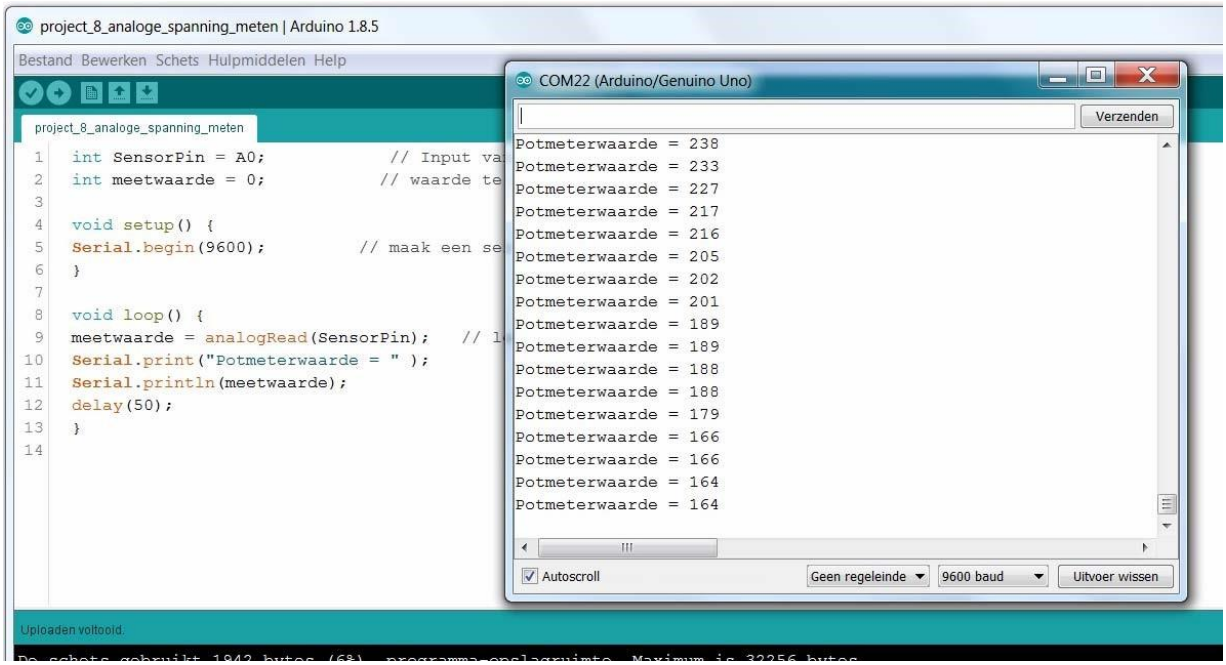
$$U_2 = U * \frac{R_2}{R_1 + R_2}$$



De meetwaarde kan Arduino “binnenhalen” met het commando analogRead(meetpin) Het weergeven van wat er gemeten is gaat weer door uitvoer naar de Seriële monitor.

```
 int SensorPin = A0;   // Input van de potmeter
 int meetwaarde = 0;   // waarde te meten van A0 noemen we potmeter

 void setup() {
 Serial.begin(9600);   // maak een seriële communicatie op 9600 bps:
 }

 void loop() {   meetwaarde = analogRead(SensorPin);  // lees de
analoge waarde:
 Serial.print("Potmeterwaarde = " );
 Serial.println(meetwaarde);
delay(50);                    }
```

Nadat de Seriële Verbinding op de laptop gestart is wordt voortdurend de waarde van de potmeter op het scherm weergegeven. Als je de potmeter verdraait zie je nu op de laptop een getal tussen de 0 en de 1023 Een waarde 0 komt overeen met 0 volt, de waarde 1023 komt overeen met 5 volt



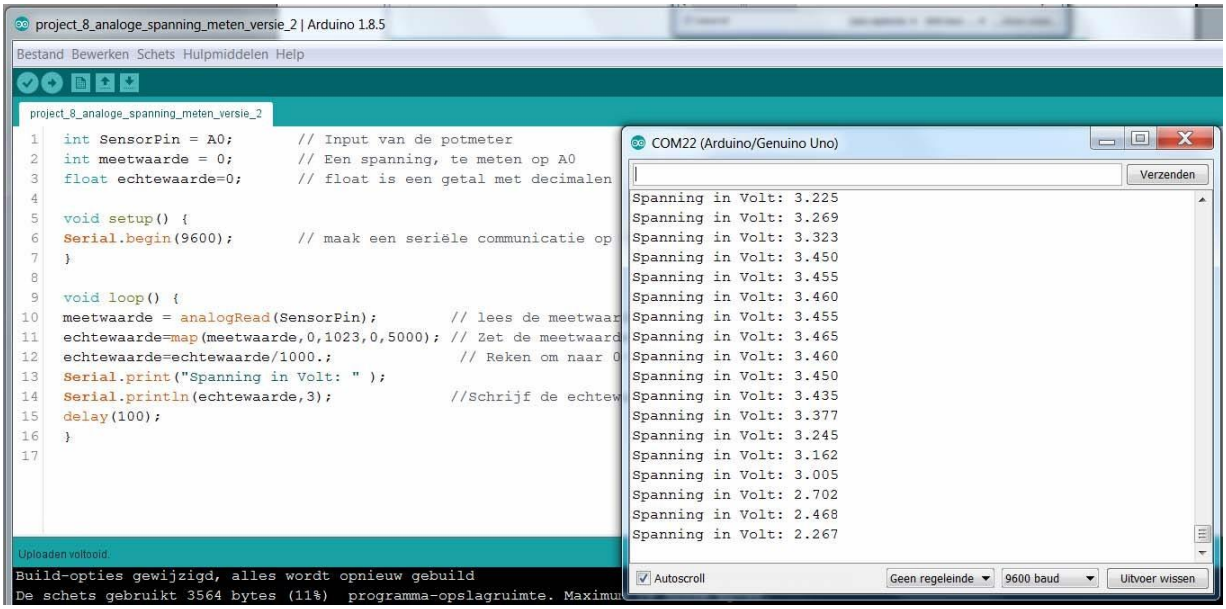
Maar we willen geen waarde zien van 0 tot 1023 maar de echte waarde.
Arduino zet de gemeten spanning op een Analoge pin om naar een getal tussen de 0 en 1023. Als we echt een meetwaarde van 0 tot 5 Volt op de Seriele Monitor willen zien moeten we de gemeten waarde van 0-1023 'mappen' naar een waarde tussen de 0 en 5. Daar is een handig commando voor met de naam *map*. Dat zet een ingangswaarde in de range fromLow-fromHigh om naar een waarde in de range toLow-toHigh. Omdat map alleen werkt met hele getallen (integers) zetten we de meetwaarde eerst om naar een getal tussen de 0 en 5000. Daarna delen we die uitkomst door 1000.

```
 int SensorPin = A0;      // Input van de potmeter   int
meetwaarde = 0;     // Een spanning, te meten op A0   float
echtewaarde=0;   // float is een getal met decimalen

 void setup() {
 Serial.begin(9600);                          // maak een seriële communicatie op 9600 bps:
 }

 void loop() {   meetwaarde = analogRead(SensorPin);                 // lees de meetwaarde:
echtewaarde=map(meetwaarde, 0, 1023, 0, 5000); // Zet de meetwaarde om naar iets tussen de 0 en 5000
echtewaarde=echtewaarde/1000.;                 // Reken om naar 0-5 Volt door te delen door 1000
Serial.print("Spanning in Volt: " );
 Serial.println(echtewaarde,3);           //Schrijf de echtewaarde in 3 decimalen naar de Seriële monitor
delay(100);                    }
```

In dit programma wordt een nieuw soort variabele gebruikt, float float is de korte naam voor een 'floating-point getal. Getallen dus met een decimale punt.
Ideaal als het gaat om meetwaarden als spanning, temperatuur enz.

Ook het commando Serial.println(echtewaarde,3); is in deze vorm nieuw. Deze opdracht geeft de waarde op de seriële monitor in 3 decimalen.

Het resultaat is



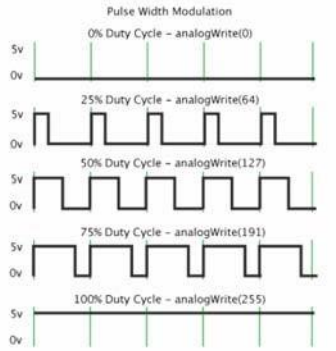
Je kunt vaak regels in programma's combineren. Dat maakt het programma korter (maar niet altijd beter leesbaar….)

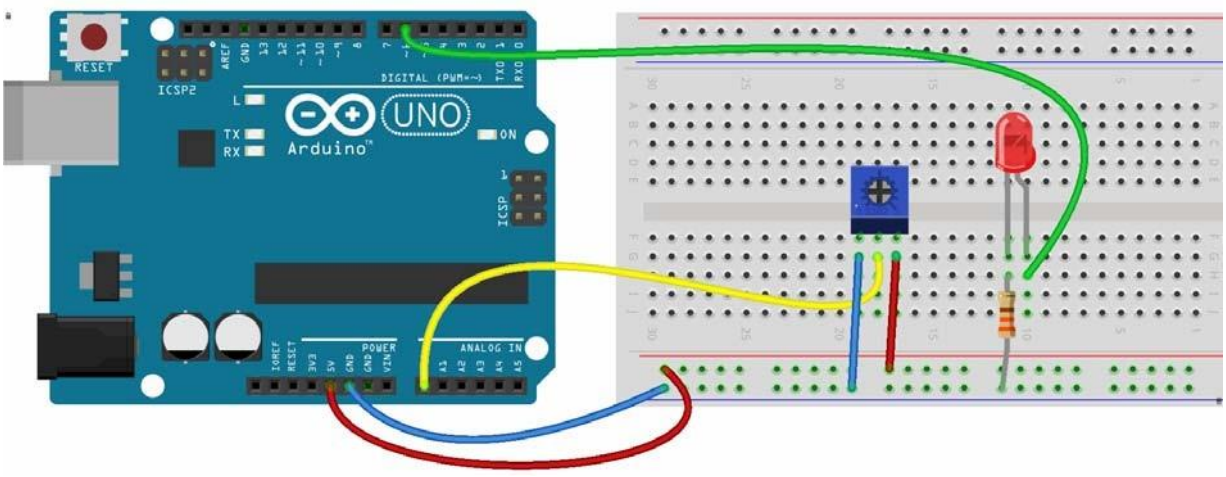
```
void loop() {   echtewaarde=map(analogRead(SensorPin), 0, 1023, 0, 5000) /1000;     // meten, mappen en
delen in één!
  Serial.print("Spanning in Volt: " );
  Serial.println(echtewaarde,3);            //Schrijf de echtewaarde in 3 decimalen naar de Seriële monitor
delay(100);                    }
```

Project 11: Een LED dimmen

LED's branden of branden niet. Maar door ze heel snel aan en uit te zetten lijken ze gedimd.
Dat snel aan/uit schakelen gaat het makkelijkste met de speciale Digitaloutputs pen 3,5,6,9,10 en 11.
(Deze pinnen hebben een ~ voor het nummer)
Met deze pennen is "*PulseWidthModulation*" mogelijk. Met een analogWrite commando naar deze pinnen wisselt daarna de toestand van zo'n pin van hoog naar laag in een bepaald tempo.
Bij analogWrite(255) staat de pin voortdurend aan, bij analogWrite(127) maar voor de helft.
Een LED die is aangesloten op deze pinnen kun je dus dimmen!
Dan moeten we deze keer die analoge waarde van 0-1023 dus *mappen* naar een waarde tussen de 0-255. Die waarde sturen we daarna met analogWrite naar de PWM pin waarop de LED is aangesloten.

Het resultaat: een LED die gedimd kan worden!





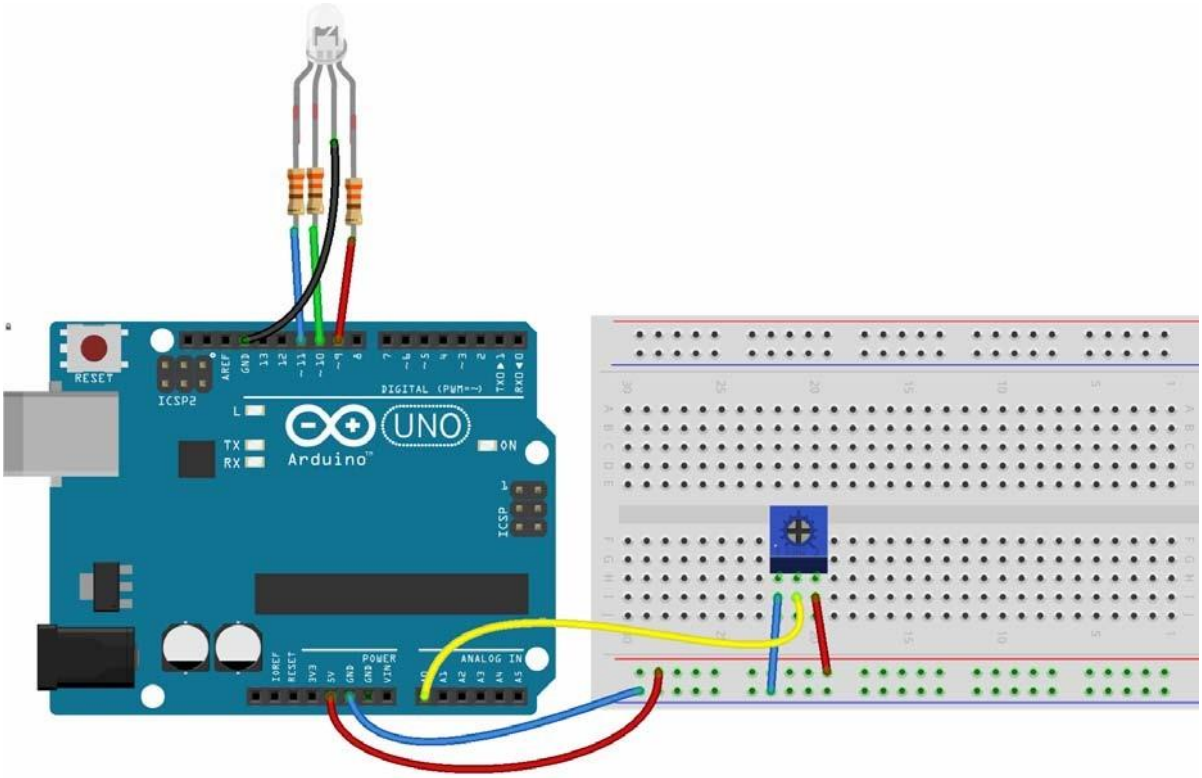

```
int ledPin =  6;       // the number of the LED pin int
SensorPin = A0;      // Input van de potmeter int
meetwaarde = 0;      // Een spanning, te meten op A0 int
PWMwaarde = 0;

void setup() {
Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() {   meetwaarde = analogRead(SensorPin);       // lees de
meetwaarde:
  PWMwaarde = map(meetwaarde, 0, 1023, 0, 255); // Zet de meetwaarde om naar iets tussen de 0 en 5000
analogWrite(6, PWMwaarde);
  delay(100);
}
```

Project 12: RGB Led's

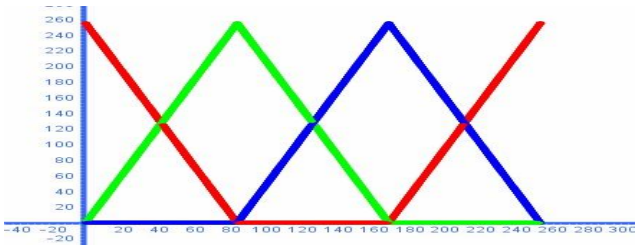
Het wordt nog leuker als we een RGB LED gebruiken. Dat zijn eigenlijk 3 LED's in één huisje.
Een Rode LED, een Groene LED en een Blauwe LED. Door van elke LED de intensiteit te varieren kun je alle kleuren van de regenboog maken.

Aansluitingen:

Let op! Dit is een "Common Kathode" RGB ! (Er zijn er ook met eeg "Common Anode") We moeten PWM pinnen gebruiken. 9, 10 en 11 liggen mooi bij elkaar. En daar is ook een Gnd aanwezig.



Elke led kan precies als voorheen met digitalWrite aan en uitgezet worden én met analogWrite gedimd worden. We gebruiken een potmeter om een waarde van 0-255 in te stellen. (Net als in het vorige project) Maar deze keer berekenen we uit die waarde de kleurwaarde waarmee we de Rode, Grone en Blauwe led moeten aansturen. Zo krijgen we een instelknop waarmee we elke ledkleur kunnen instellen. Die berekening is in een functie gestopt. Er "in" gaat een waarde van 0 tot 255, er "uit" komt een Rwaarde, een G-waarde en een B-waarde waarmee de led's aangestuurd gaan worden.
Met een beetje gepuzzel is te zien dat Conversion(int ColorValue) inderdaad dit oplevert.
Een waarde 0 levert 100% rood op. Een waarde 255 óók.
Als we met blauw willen eindigen zouden we in *map* ook kunnen begrenzen op 0 – 170.



Het programma:

```
int SensorPin = A0;       // Input van de potmeter int
meetwaarde = 0;       // Een spanning, te meten op A0 int
PWMwaarde = 0;

int RedLed = 9; int
GreenLed = 10;
int BlueLed = 11;

int R = 0; int
G = 0;
int B = 0;

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

void loop() {    meetwaarde = analogRead(SensorPin);        // lees de
meetwaarde:
  PWMwaarde = map(meetwaarde, 0, 1023, 0, 255); // Zet de meetwaarde om naar iets tussen de 0 en 255 (of 170)
  Conversion(PWMwaarde);
analogWrite(RedLed, R);   analogWrite(GreenLed,
G);   analogWrite(BlueLed, B);
  delay(100);
}

int Conversion(int ColorValue) {
if (ColorValue < 85) {     R = 255
- ColorValue * 3;
    G = 0;
    B = ColorValue * 3;
  }
  else if (ColorValue < 170) {
ColorValue = ColorValue - 85;
    R = 0;
    G = ColorValue * 3;
    B = 255 - ColorValue * 3;
  }
else {
    ColorValue = ColorValue - 170;
    R = ColorValue * 3;
    G = 255 - ColorValue * 3;
    B = 0;
  }
}
```

Project 13: Arrays

Als we drie (of nog meer) LED's aan willen sluiten en achter elkaar als een 'looplichtje' willen laten knipperen moeten we voor elke LED hetzelfde stukje code schrijven. Bijvoorbeeld…

```
int LedPin1=10; int
LedPin2=11; int
LedPin3=12;
enzovoort

void setup() {
  pinMode(LedPin1, OUTPUT);
pinMode(LedPin2, OUTPUT);   pinMode(LedPin3,
OUTPUT);
  enzovoort
}

void loop() {
  digitalWrite(LedPin1, HIGH);
  delay(50);
  digitalWrite(LedPin1, LOW);
  delay(1000);

  digitalWrite(LedPin2, HIGH);
  delay(50);
  digitalWrite(LedPin2, LOW);
  delay(1000);

  digitalWrite(LedPin3, HIGH);
  delay(50);
  digitalWrite(LedPin3, LOW);
delay(1000);
   enzovoort

 }
```

Dat is onhandig. Wat we eigenlijk willen programmeren is iets als "*doe eerst iets met de LED op pin 10, daarna hetzelfde met de LED pin 11 en daarna hetzelfde met de LED pin 12*."
We willen dus drie keer achter elkaar hetzelfde doen maar het enige verschil is het pinnummer.
We hebben twee dingen nodig.

I. Een 'rijtje' met de ledpin nummers.
Een rijtje heet in programmeertaal een '*array*'.
Met de volgende opdracht vertellen we de Arduino dat er in de variabele *ledpins* drie ledpinnummers staan namelijk nr. 10, 11 en 12 `int ledpins[3]={10,11,12};`
Als we nu in ons programma ergens `digitalWrite(ledpin[2], HIGH);` schrijven dan zou de led op pin 11 aan moeten zetten. Dat klopt bijna maar niet helemaal.
In Arduino wordt de *eerste* in een array met `ledpin[0]` aangegeven, de *tweede* met `ledpin[1]` enzovoort.
Als we de drie led's op pin 10,11 en 12 aan willen zetten moeten we dus schrijven `digitalWrite(ledpin[0], HIGH);` `digitalWrite(ledpin[1], HIGH);` `digitalWrite(ledpin[2], HIGH);`

Maar nu zijn we toch nog steeds bezig om steeds hetzelfde stukje code voor elke Led apart te schrijven? Klopt!

II. Daarom hebben we een 'tellertje' nodig dat van 0 tot en met 2 doortelt. Tellertjes komen heel veel voor in programma's. Het gaat zo…

```
for (i=0;  i<3;  i++)
{
   pinMode(ledpins[i],OUTPUT);
}
```

In gewone taal staat hier:
a. Geef de variabele *i* eerst de waarde 0.
b. Doe alles wat er tussen de accolades staat.
c. Verhoog daarna de teller met 1. Die opdracht kan in Arduino heel kort. i++
d. Controleer of de teller nog geen 3 is.
Nee? Blijf dan de stappen b t/m d herhalen.
Ja? Dan zijn we klaar!

Hiermee kan ons programma om drie LED's na elkaar te laten knipperen een stuk korter.

```
/* Project 12. Meerdere led's en het gebruik van een array */ int
ledpins[3] = {10, 11, 12};

void setup() {
  for (int i = 0;  i < 3;  i++) {
    pinMode(ledpins[i], OUTPUT);
  } }
void loop() {
  for (int i = 0;  i < 3;  i++) {
digitalWrite(ledpins[i], HIGH);     delay(50);
    digitalWrite(ledpins[i], LOW);
    delay(100);
  }
}
```

Project 14: Een intelligente schakelaar met Debouncer

In project 6 deden we iets dat onmogelijk zonder Arduino had gekund. De Arduino en het programma maakte onze schakelaar "*intelligent*". Eenmaal drukken en de LED flitst 3x op.
Nóg een voorbeeld van een slimme schakelaar is een "*omschakelaar*".We willen dat:

- de LED aangaat als je één keer op de schakelaar drukt.
- de LED uitgaat als je nóg een keer op de schakelaar drukt.

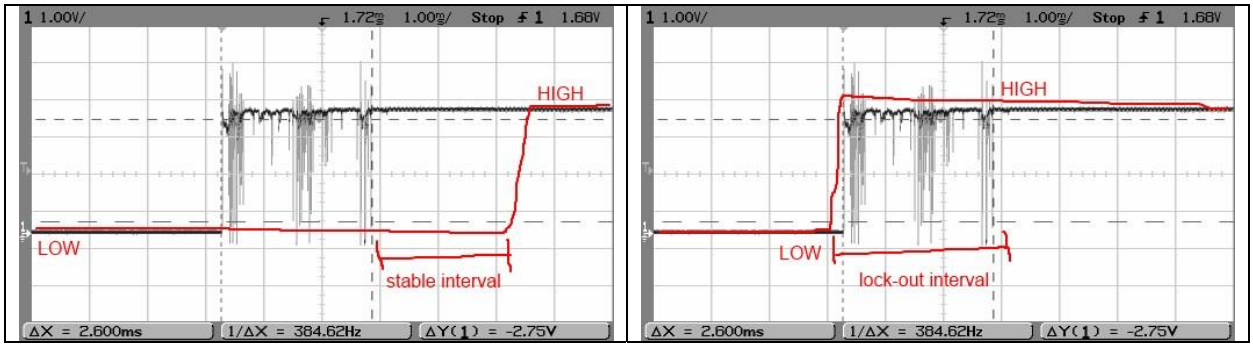
Maar, als er iets de ene keer iets ánders moet gebeuren dan de ándere keer dat je op de schakelaar dukt, dan móet het programma onthouden wat 'de vorige toestand' was.
We moeten de *toestand* van de schakelaar onthouden in een variabele *lastbuttonstate* .
Arduino kan dan beslissen dat de schakelaar wordt ingedrukt als we de schakelaar 'inlezen' met digitalRead en vergelijken met de vorige toestand. Maar we moeten ook onthouden of de LED aan was of juist uit. Als-ie aan stond moet-ie nu uit, als-ie uit stond moet-ie juist aan!
We gebruiken een variabele *ledstate* om de toestand van de led te onthouden. Een stukje van het programma.

```
void loop()
{   buttonstate = digitalRead(buttonpin);   if (buttonstate == HIGH && lastbuttonstate == LOW)
{ // Dan wordt nu de button ingedrukt.

   if (ledstate == HIGH) {  // Als de Led aan staat, zet 'm dan uit.
digitalWrite(ledpin, LOW);       ledstate = LOW;
   }     else {                        // Als de Led uit staat, zet 'm dan
aan.
     digitalWrite(ledpin, HIGH);
ledstate = HIGH;
   }
}
  lastbuttonstate = buttonstate; }
```

Het vorige programma kan veel korter. We willen dat de led áán gaat als-ie uit staat en omgekeerd. De *ledstate* moet dus 'omgekeerd' worden. Daar is een korte opdracht voor: ledState = !ledState; En dan wordt ons programma veel korter en overzichtelijker!

```
void loop()
{
  buttonstate = digitalRead(buttonpin);
  if (buttonstate == HIGH && lastbuttonstate == LOW) ledstate
= !ledstate;   lastbuttonstate = buttonstate;   digitalWrite(ledpin,
ledstate);
}
```

Een mooi kort programma waarmee we een LED (in dit geval de ingebouwde LED op pin 13) aan/uit kunnen schakelen met één drukknop.
Maar als je dit programma uitprobeert merk je iets vreemd…. Het werkt soms niet.
Eigenlijk is het een wonder dat het meestal wel werkt want schakelaars zijn geen perfecte schakelaars.
Als je op de knop drukt wordt het contact niet gesloten maar het '*stuitert'*.
Het gaat wel 10 keer weer open en dicht voor het eindelijk gesloten blijft.
Maar als dat eens een keer 11 keer is denkt het programma dat de LED weer uit moet….
Dat stuiteren van het schakelcontact is mooi in beeld gebracht in het volgende plaatje.



En tegelijk staat hier een ‘strategie’: Wacht net zolang tot het gestuiter afgelopen is (of te wel dat het signaal niet meer verandert) en wacht daarna nog een bepaalde tijd en wat je dán uitleest is de juiste schakelaarpositie. (Linkerplaatje) Het kan ook anders. Zodra de schakelaar van stand verandert wacht je een “lock-out interval” en de positie op dat moment is bepalend. (Rechterplaatje) Die eerste manier is in het volgende programma opgenomen.
We gebruiken opnieuw een functie (om daarmee het programma overzichtelijk te houden).

```
const int buttonpin = 7;
const int ledpin = 13;
int ledstate = LOW; int
buttonstate;
int lastbuttonstate;

void setup()
{
 pinMode(ledpin, OUTPUT);
 pinMode(buttonpin, INPUT_PULLUP);
}

void loop()
{   buttonstate = !debounce( buttonpin); // Vanwege de Pull-up is ingedrukt keren we het om.
 if (buttonstate == HIGH && lastbuttonstate == LOW) ledstate = !ledstate;   // Button ingedrukt, dDraai ledstate om
lastbuttonstate = buttonstate; // Onthoud de vorige toestand van de button   digitalWrite(ledpin, ledstate);  // Zet de led
nu in ledstate.  }

boolean debounce(int pin)
{
 boolean state;   boolean previousState;   previousState = digitalRead(pin);  // Lees de button voor de 1e
keer in.   for (int counter = 0; counter < 10; counter++)   // Lees nu 10 msec. lang steeds opnieuw de
button in.
 {
   delay(1);                          //wait 1 millisecond     state = digitalRead(pin);        //read the pin     if
(state != previousState)      // Als in die 10 msec. De buttonstate veranderd gaat opnieuw 10 msec. In.
   {       counter = 0;                      //reset the counter
previousState = state;  // Kennelijk is de button 10 msec. stabiel.
   }
 }
 return state; // Geef dan nu aan het programma de status van de button terug.  }
```

Het mooie van dit programma is dat het eenvoudig uit te breiden is tot meerdere schakelaars. Bijvoorbeeld 3 schakelaars.

```
const int switch0 = 5; const
int switch1 = 4; const int
switch2 = 3;

void loop()
{
  pinMode(switch0, INPUT_PULLUP);   pinMode(switch1,
INPUT_PULLUP);   pinMode(switch2, INPUT_PULLUP);

  if (debounce(switch0) == HIGH) {       // Doe dan iets!
  }
  if (debounce(switch1) == HIGH) {       // Doe dan iets!
  }
  if (debounce(switch2) == HIGH) {       // Doe dan iets!
  }
}

boolean debounce(int pin)
{
  boolean state;
boolean previousState;
  previousState = digitalRead(pin);
  for (int counter = 0; counter < 10; counter++)
  {
    delay(1); //wait 1 millisecond     state
= digitalRead(pin); //read the pin     if
(state != previousState)
    {
      counter = 0; //reset the counter
previousState = state;
    }
}
  return state;
}
```

In dit programma is float een nieuw datatype gebruikt.
We kenden al de int variabele. Ee getal dat varieert van -32.768 tot 32.767 We kenden ook de float, het floating point getal.
Ook gebruikten we al eens het array.
Hier wordt de boolean gebruikt. Die kent maar twee waarden, HIGH of LOW.
Daarnaast is er nog de byte, een getal dat van -255 tot 255 kan varieren.
Als een variabele niet veel varieert, (zoals een pin nummer) zou je er ook een byte voor kunnen gebruiken. Dat scheelt een klein beetje geheugen.
Een ander belangrijke datatype is de char die voor “character” variabelen wordt gebruikt.
Bijvoorbeeld: char myChar = 'A';

Project 15: Het gebruik van een library

Een andere manier is om een aantal "elementen" van het programma (starten van een timer, detecteren of de toestand veranderd is enz.) in een bibliotheek te stoppen. Zo'n bibliotheek moet "*ge-include*" worden. De Bounce2 bibliotheek geeft de mogelijkheid om een buttonpin te *attachen* aan een functie die in dit geval Debouncefunctie is genoemd. (Maar elke naam was goed geweest…) Na enkele van zulke magische regels is daarna het statement "Debouncefunctie.read(); voldoende om de buttonpin betrouwbaar uit te lezen. We kunnen daarna met die correct_uitgelezen_schakelaar van alles doen. In dit geval wordt-ie gebruikt om een ledstate te togglen. We hebben hier dus een omschakelfunctie met één schakelaar.

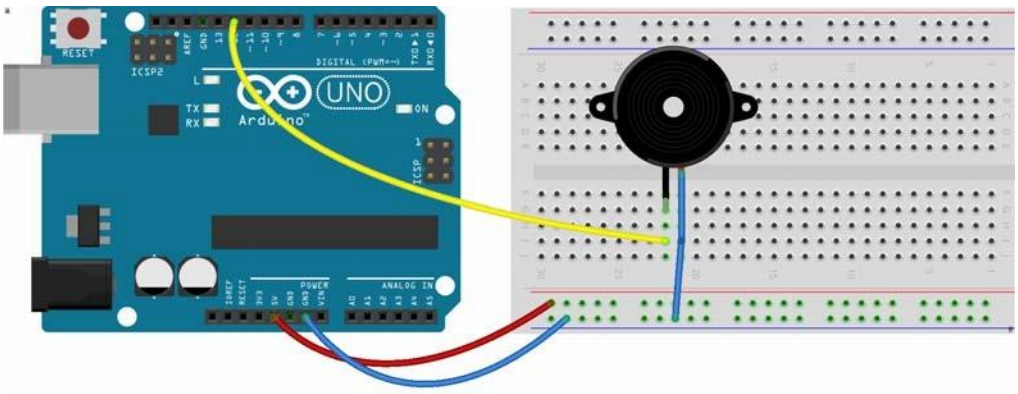
```
#include <Bounce2.h>

Bounce Debounce1 = Bounce(); // Instantiate a Bounce object

const int buttonpin = 7;
const int ledpin = 13;
int ledstate = LOW; int
buttonstate;
int lastbuttonstate;

void setup()  {
  pinMode(ledpin, OUTPUT);
pinMode(buttonpin, INPUT_PULLUP);
  Debounce1.attach(buttonpin);          // After setting up the button, setup the Bounce instance :
Debounce1.interval(5);                  // interval in ms }

void loop()  {
  Debounce1.update();                                                    // Update the Bounce instance :
buttonstate = !Debounce1.read();                                       // Get the updated value :
  if (buttonstate == HIGH && lastbuttonstate == LOW) ledstate = !ledstate;  // Toggle de ledstate
digitalWrite(ledpin, ledstate);   lastbuttonstate = buttonstate; }
```

Bij vrijwel elke library worden voorbeelden meegeleverd die inspiratie en instructie zijn hoe die library gebruikt moet worden. In Voorbeelden, Bounce2 staat de voorbeeld sketch bounce2buttons De volgende essentiele regels (geen volledige sketch) laten de uitbreiding zien tot 2 buttons.

```
Bounce debouncer1 = Bounce(); // Instantiate a Bounce object
Bounce debouncer2 = Bounce(); // Instantiate another Bounce object

void setup() {
  debouncer1.attach(Buttonpin_1);   debouncer1.interval(5);
// interval in ms   debouncer2.attach(Buttonpin_2);
debouncer2.interval(5); // interval in ms
}

void loop() {
debouncer1.update();
debouncer2.update();

  int value1 = debouncer1.read();
int value2 = debouncer2.read(); }
```

Project 16: Audio

Piezobuzzers kunnen met een uitgang en de + maar net zo goed met de min verbonden worden. Ze hebben niet -zoals Ledjes- zelf ook een + en -)



Een buzzer is een miniatuurluidspreker. Niet zo'n hele goeie maar voor wat simpel gepiep doet-ie het prima.
In het volgende programma gebruiken we de opdracht Tone.
In die opdracht moeten we drie dingen opgeven. Buzzerpin, toonhoogte (frequentie) en hoe lang die toon moet klinken.
Na elke Tone opdracht komt ook een delay.
Om het 'componeren' iets makkelijker te maken is al iets gemaakt voor het verschil tussen een kwart noot en een achtste noot. tijdsduur/4 is een kwart noot, tijdsduur/8 is een achtste noot. Als je één keer de variabele *tijdsduur* aanpast gaat alles sneller of juist langzamer. (Probeer eens tijdsduur=450;)

```
int buzzerpin=12;
int tijdsduur=750;

void setup () {                  // Hier staat helemaal niets...
}

void loop(){

  tone(buzzerpin, 392,tijdsduur/4); //G4   delay(1.3*tijdsduur/4);
  tone(buzzerpin, 523,tijdsduur/4); //C5    delay(1.3*tijdsduur/8);
  tone(buzzerpin, 659,tijdsduur/4); //E5   delay(1.3*tijdsduur/8);
  tone(buzzerpin, 784,tijdsduur/3); //G5   delay(1.3*tijdsduur/4);
  tone(buzzerpin, 659,tijdsduur/4); //E5   delay(1.3*tijdsduur/4);
  tone(buzzerpin, 784,tijdsduur/1); //G5
delay(1.3*tijdsduur/1); }
```

Maar we hebben net gemerkt dat een programma veel overzichtelijker wordt als we bij elkaar horende regels in een "Functieblok" stoppen. Dan kunnen we hier ook doen.

```
int buzzerpin = 12;
int tijdsduur = 850;

void setup () { // Hier staat helemaal niets... }

void loop() {
melodietje();
}

void melodietje() {
  tone(buzzerpin, 392, tijdsduur / 4); //G4
delay(1.3 * tijdsduur / 4);
  tone(buzzerpin, 523, tijdsduur / 4); //C5
delay(1.3 * tijdsduur / 8);   en de rest….
}
```

Om zo een melodietje te schrijven is lastig want muziek werkt niet met frequenties maar met tonen. Een toon van 440 hz. is een A. (Eigenlijk de vierde op de piano dus A4 In plaats van iets vaags als tone(buzzerpin, 440, tijdsduur / 4); //A4 zou je eigenlijk willen schrijven tone(buzzerpin, A4, tijdsduur / 4);
(en dan natuurlijk van te voren definiëren int A4=440;

En dat dan voor alle tonen. Dat is al eens uitgezocht!
Een klein stukje uit die lijst… Hiermee zou het programma moeten beginnen….

```
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
```

(Waarom niet int NOTE_A3=220; ?
Nou, deze waarden worden nooit in het programma gewijzigd.
Dan is #define NOTE_A3 220 (zonder de ; !!) net iets beter want dat kost minder geheugenruimte.)

Project 17: Het ‘Include’ statement

Maar met zo’n lange lijst van 88 tonen wordt het programma weer onoverzichtelijk.
Er is een mooie manier om dat op te lossen. Maak in Arduino een extra tabblad aan! Rechtsboven maar niet helemaal boven, net daaronder….

Maak een nieuw tabblad aan en geef dat de naam pitches.h.
Op dat nieuwe tabblad komt de hele lijst van tonen en frequenties te staan. (Je krijgt die lijst…)
We moeten Arduino nog wel vertellen dat daar die tonen staan. Dat gaat met de opdracht #include "pitches.h"

sketch_may10a § pitches.h §

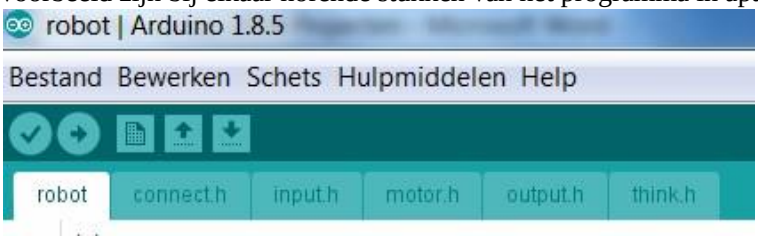
```
int buzzerpin = 12;
int tijdsduur = 750;

#include "pitches.h" // Op dat tabblad worden alle namen en freqenties gedefinieerd.

void setup () {
}

void loop()
{
  melodietje();
}

void  melodietje() {
  tone(buzzerpin, NOTE_G4, tijdsduur / 4); //G4
  delay(1.3 * tijdsduur / 4);
  tone(buzzerpin, NOTE_C5, tijdsduur / 4); //C5
  delay(1.3 * tijdsduur / 8);
  tone(buzzerpin, NOTE_E5, tijdsduur / 4); //E5
  delay(1.3 * tijdsduur / 8);
  tone(buzzerpin, NOTE_G5, tijdsduur / 3); //G5
  delay(1.3 * tijdsduur / 4);
```

Als een extra tabblad wordt uitgemaakt zónder die “.*h*” in de naam is het voor Arduino net als een volgende bladzij van de sketch. Alle bladzijden worden achter elkaar gelezen. Dus als er alleen functieblokken staan op een tabblad “Functieblok” is er geen #include Functieblok nodig.
Maar als er definities van variabelen staan is het wel belangrijk dat die vooraan in het programma worden geplaatst. #include "pitches.h" forceert dat dat hele blok met definites van tonen aan het begin van het programma wordt “ge-include”.

Let op het verschil!

- In project 14 hebben we een library ge-include met #include <Bounce2.h>
  Die library moest op een bepaalde plek staan namelijk in C:\Program Files\Arduino\libraries óf, beter eigenlijk- in C:\users\usernaam\Documents\Arduino. Na het wijzigen of plaatsen van een library moet Arduino opnieuw gestart worden!

- #include "pitches.h" leest op de plaats van die regel het tabblad pitches.h in. Dat moet in dezelfde map staan waar de Arduino sketch staat. Je kunt ook meerdere tabbladen gebruiken. In dit voorbeeld zijn bij elkaar horende stukken van het programma in aparte tabbladen ondergebracht.




```
#include "connect.h"
#include "motor.h"
#include "input.h" #include
"think.h"
#include "output.h"
```

Project 18: Maak eens een leuke deurbel!

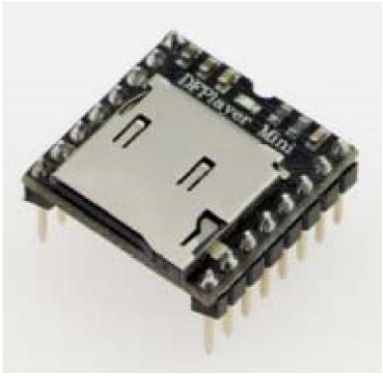
Schrijf een programma en bouw een schakeling met een buzzer en een schakelaar.
Programmeer het zó dat als je op de knop drukt, er drie keer een melodietje wordt gespeeld.

Variantie voor later (studentenhuis…)
Maak 2 of 3 drukknoppen en zorg dat elke knop z'n eigen melodietje heeft.

Op de USB stick zijn een aantal sounds sketches opgenomen.
Zoals:
- *Arduino_Sound_Effects*
- *Fur_Elise*
- *JingleBells*
- *Pirates_of_the_Caribbean_Theme_Song*
- *R2_D2_sound*
- *Super_Mario*
- *Twinkle_Twinkle_Little_Star*
- *PlayMelody_1, 2 en 3*
- *Harry Potter Theme*

Het klinkt eigenlijk nergens naar. Het probleem is dat Arduino voor tonen z'n timers nodig heeft. Zoals wel eens gezegd wordt: "The Arduino is an amazing platform for all kinds of projects, but when it comes to generating sound, many users struggle to get beyond simple beeps. "
Kan het ook anders? Niet zo eenvoudig…

Er zijn shields te koop die de afhandeling van sounds voor hun rekening nemen, en daarmee de Arduino ontlasten. B.v. het DFPlayer_Mini shield van DFrobot.com.
Dit is een interessant –en goedkoop- shield met mogelijkheden. De sounds worden als .WAV bestand op een memorycard geplaatst. De software geeft mogelijkheden voor volume, nummerkeuze enz. enz.

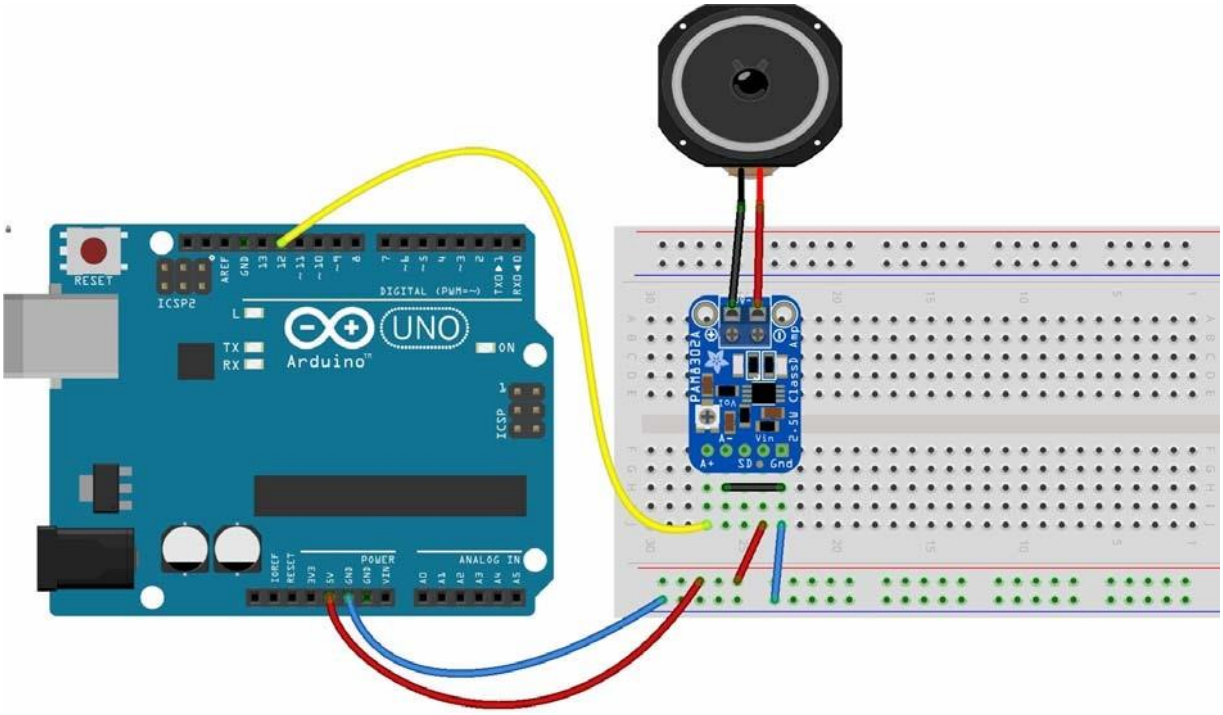


https://www.dfrobot.com/wiki/index.php/DFPlayer_Mini_SKU:DFR0299

Project 19: Meer volume met een class D versterker

Zo'n buzzer maakt nauwelijks geluid.
Maar ipv op de buzzerpin een buzzer aan te sluiten kunnen we net zo goed een kleine versterker aansluiten.
In dit geval een "Adafruit Class D amplifier", gebaseerd op een PAM 8302A chip.
Er is een instelpotmeter voor het volume aanwezig. Die is nodig ook…. Dit
maakt echt geluid!

Deze amplifer heeft zgn "differential inputs A+ en A-". Maar de A- wordt verbonden (is aan de onderzijde al verbonden !) met Gnd. Dus alleen A+ is hier ingang.



Zie verder https://www.adafruit.com/product/2130

Opmerkingen over programmeren van Arduino.

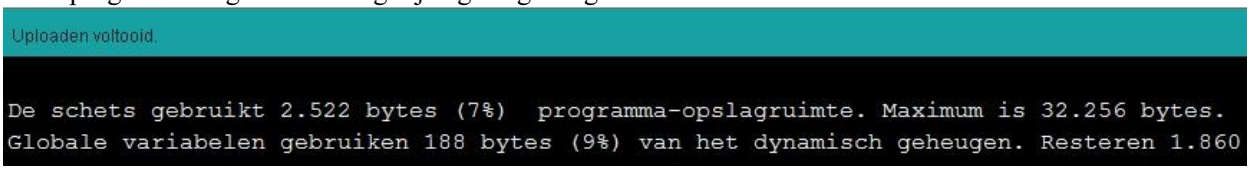
1. Arduino is Hoofd en kleine letter 'gevoelig'.
   Er komen allerlei vreemde foutmeldingen als je daar een foutje in maakt.
   Arduino weet niet meer over welke variabele je het hebt als je een hoofd ipv kleine letter gebruikt.

2. Er komen ook vreemde foutmeldingen als je ergens een ; bent vergeten. De beste oplossing is:
   Vergeet niet om elke regel met ; af te sluiten!

3. Ook krijg je vreemde foutmeldingen als je ergens een { of } teveel of te weinig hebt.
   Arduino is geen heel slimme computertaal die de fout voor jou opzoekt! Het kan vaak heel lastig zijn om de ontbrekende accolade te vinden.
   Eén manier is om de "opmaak-sneltoets" **Ctrl-T** te gebruiken.
   Als je ergens een } vergeten bent zie je dat vaak dan wat sneller.
   Een andere manier is om je programma in blokjes op te delen en langzaam uit te breiden.

4. Verschillende variabelen zijn als int 'gedeclareerd'. (Zo heet het als je een variabele een naam en een type geeft..)

Maar b.v. zo'n ledpin is altijd iets tussen de 0 en de maximaal 13. Dat kunnen we dan net zo goed als een byte declareren. Wat levert dat op? Geheugen! Daarvan heeft een Arduino niet zoveel.
Na het uploaden zie je iets als hieronder.
Onze programma's gebruiken nog bijna geen geheugen.

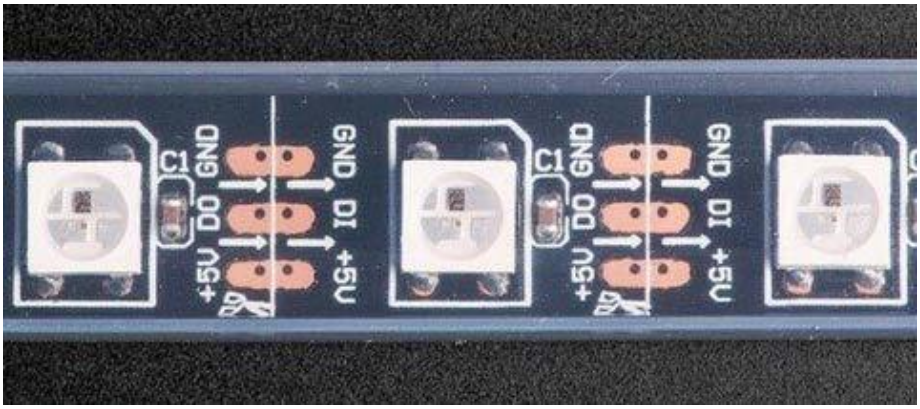


Maar bij grote programma's moet je echt rekening houden met het soort variabelen en de ruimte die nodig is.

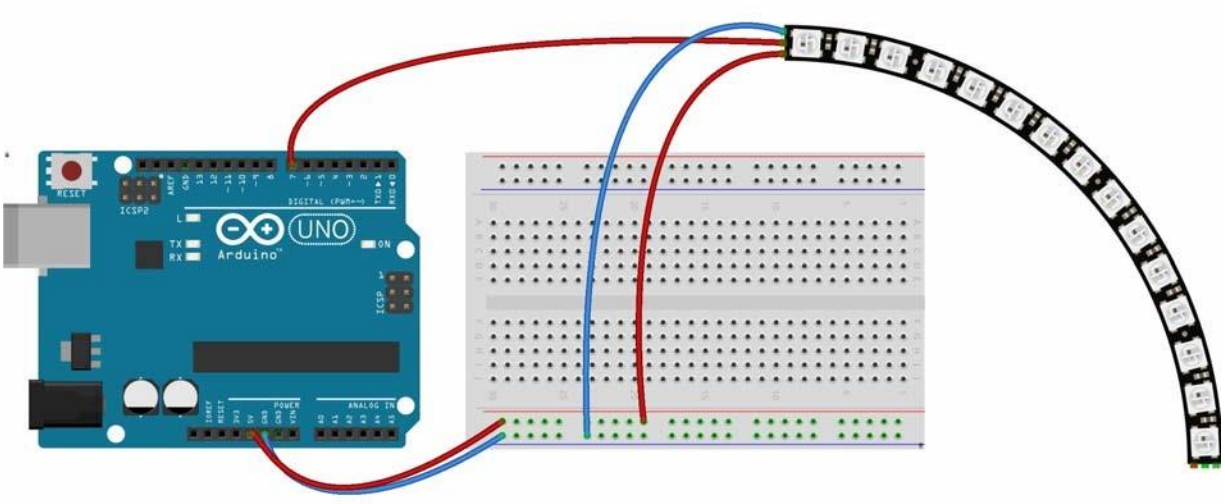
Overigens is dit niet helemaal waar. De Analoge pinnen A0 t/m A5 kúnnen ook als digitale pin worden gebruikt. Ze nummeren dan gewoon door als 14 t/m 19

Project 20: Heel veel LED's in een Neopixel strip

Een Neopixel is een combinatie van een Rode, Groene en Blauwe LED in één behuizing. Maar er is tegelijkertijd een stukje electronica mee geïntegreerd. Elke LED heeft z'n eigen kleine chipje. En daardoor kunnen alle LED's achter elkaar geschakeld worden want elke LED heeft zijn eigen adres. Het resultaat: een lange strip LED's waarvan elke LED in de gewenste kleur kan worden gezet terwijl er niets anders nodig is als een +, een – en een Data draad.



Om zo'n Neopixels strip vanuit een Arduino aan te sturen moeten we precies die aansluitingen maken.
Een + (rood), een – draad (zwart) en de Datadraad (bruin).
Omdat Neopixels veel stroom vragen is het handig om een dikke condensator op de voeding te solderen.
Dat is hier al gedaan. Verder is een netvoeding (max 12V) aan te raden. Een 9V batterij is snel leeg.
En een laptop USB uitgang kan niet veel stroom leveren.
Om de Neopixels te kunnen aansturen is de library "*Adafruit_NeoPixel-master*" nodig.



Verbind Gnd met Gnd, Vcc met 5V op de Arduino en de Data lijn op pin 7. Het aantal Neopixels in deze sliert is 72.

Een stukje van de sketch.
In het programma volgen de volgende functies elkaar op met steeds 2 seconden er tussenin.

```
  colorWipe(Ledstrip.Color(255, 255, 255), 50
  theaterChase(Ledstrip.Color(127, 127, 127), 100); // White
singlePixel_Heen(Ledstrip.Color(255, 255, 0), 10); //lopend lichtje
singlePixel_Terug(Ledstrip.Color(255, 255, 0), 10); //lopend lichtje
  All_Pixel_On(Ledstrip.Color(255,  255,  255),  500);
All_Pixel_On(Ledstrip.Color(0    ,  0    ,  0    ),  500);
rainbow(50);                              rainbowCycle(50);
theaterChaseRainbow(50);
```

Op de Seriële Monitor is steeds te zien welke functie aan het werk is.
Zelf nieuwe functies te schrijven is niet zo eenvoudig. Maar het wijzigen van tempo of kleuren is niet moeilijk. Neopixels zijn vrij traag. De verversingsfrequentie is ongeveer 400 Hz.
Sneller (maar duurder) zijn DotStart LED strips.
Meer informatie is te vinden op https://learn.adafruit.com/adafruit-neopixel-uberguide

Een stukje van het programma….

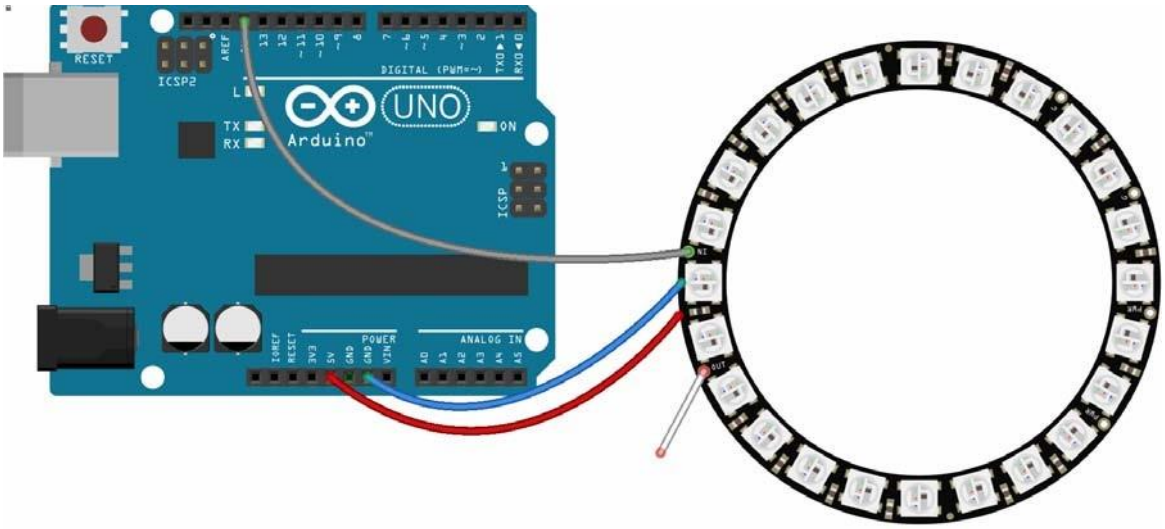
```
#include <Adafruit_NeoPixel.h>

#define Pixelpin    7    // Digital IO pin connected to the NeoPixels.

#define Numpixels 72

// Parameter 1 = number of pixels in strip,  neopixel stick has 8
// Parameter 2 = pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_RGB     Pixels are wired for RGB bitstream
//   NEO_GRB     Pixels are wired for GRB bitstream, correct for neopixel stick
//   NEO_KHZ400  400 KHz bitstream (e.g. FLORA pixels)
//   NEO_KHZ800  800 KHz bitstream (e.g. High Density LED strip), correct for neopixel stick

Adafruit_NeoPixel Ledstrip = Adafruit_NeoPixel(Numpixels, Pixelpin, NEO_GRB + NEO_KHZ800);  //
Dit is nodig om de Ledstrip functies aan te zetten.

void setup() {
Ledstrip.begin();
  Ledstrip.show(); // Initialize all pixels to 'off'
  Serial.begin(9600);
}

void loop() {//Er worden allerlei routines achter elkaar aangeroepen.
  Ledstrip.setBrightness(32);                          // Op Brightness 255 gaat de 9 V batterij érg snel leeg....

  Serial.println("Alles uit");
  colorWipe(Ledstrip.Color(0, 0, 0), 50);              // Black, alles uit. 100 msec lang   Serial.println("Rood,
Groen, Blauw en wit looplicht. Led's blijven aan");   colorWipe(Ledstrip.Color(255, 0, 0), 50);            // Red
lopend lichtje, ledjes blijven achtereenvolgens aan.   colorWipe(Ledstrip.Color(0, 255, 0), 50);          // Green
lopend lichtje, ledjes blijven achtereenvolgens aan.   colorWipe(Ledstrip.Color(0, 0, 255), 50);          // Blue
lopend lichtje, ledjes blijven achtereenvolgens aan.   colorWipe(Ledstrip.Color(255, 255, 255), 50);    // White
lopend lichtje, ledjes blijven achtereenvolgens aan.
```

Project 21: Neopixels Ring

Neopixels kunnen ook in een ring gemonteerd worden.

	
Een ring met 16 LED's	Een ring met heel veel LED's

Aansluitingen: Gnd naar Gnd, Vcc naar 5V Vcc en de Data In (grijs) naar pin 12. (Of desgewenst een andere…)



Het programma is lastig!
Een paar opmerkingen:

- De variabele Moving bepaald of de regenboog stilstaat of rondloopt.
- In de functie Rainbow wordt een kleur toegekend aan een pixelnummer. Omdat de ring symetrisch wordt gekleur hoeven we maar tot de helft van NumPixels een kleur te berekenen. Die kleurberekening gaat via het *map* kommando. Het pixelnummer wordt ge*mapt* naar een kleurwaarde tussen de 8 en de 157. (Niet van 0-255 want onder 8 staat een Led uit en boven de 157 is-ie altijd blauw/groen.)
- Uit die kleurwaarde wordt de waarde van Red, Green en Blue berekend met de functie Wheel.

Deze sketch laat 24 LED's in een ring lopen als "regenboog".
De regenboog kan stilstaan of 'lopen'.

```
#include <Adafruit_NeoPixel.h>

#define Pixelpin  12    // Grijs = Din, Wit = Dout die verbonden kan worden met de Din van een volgende ring #define
Numpixels 24     // Er zijn 24 pixels.
boolean Moving;

Adafruit_NeoPixel Ledstrip = Adafruit_NeoPixel(Numpixels, Pixelpin, NEO_GRB + NEO_KHZ800);

void setup() {
  Moving = LOW; // Als Moving=LOW, dan stilstaande regenboog, Moving=HIGH een lopende regenboog.
Ledstrip.setBrightness(10); // 10 is een mooie waarde. 64 is veel te fel   Ledstrip.begin();
  Ledstrip.show(); // Initialize all pixels to 'off' }

void loop() {
  for (int k = 0; Numpixels; k++)  rainbow(k); }

void rainbow(int k) {
  for (int i = 0; i <= Numpixels / 2; i++) {
    //Bereken een kleurwaarde voor elke piel tot de helft van NumPixels     int
kleur = map(i, 0, Numpixels / 2, 8, 157); //Niet van 0-255 maar van 8-157!     int
LedColor = Wheel(kleur);

    int LednummerLeft = (i + Moving * k) % Numpixels; // % numPixels betekent Modulo rekenen..
int LednummerRight = (Numpixels - i + Moving * k) % Numpixels;
    Ledstrip.setPixelColor(LednummerLeft, LedColor);
    Ledstrip.setPixelColor(LednummerRight, LedColor);
  }
  int LedRednummer = k % Numpixels; //% numPixels betekent Modulo rekenen. Plaats een Rode Led aan.
Ledstrip.setPixelColor(LedRednummer, 255, 0, 0);   Ledstrip.show();   delay(50); // Veel sneller is niet mooi.
}

int Wheel(int WheelPos) { // Input a value to get a R G B color value returned.   WheelPos
= 255 - WheelPos;
  if (WheelPos < 85) {
    return Ledstrip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  else if (WheelPos < 170) {
WheelPos = WheelPos - 85;
    return Ledstrip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  else {
    WheelPos = WheelPos - 170;
    return Ledstrip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  }
}
```

Project 22: Neopixels Cross

Omdat de vorige sketch toch lastig te begrijpen is volgt nu een simpeler voorbeeld.
Vier rode Led's die "rondlopen".
Aansluitingen: Gnd naar Gnd, Vcc naar 5V Vcc en de Data In naar pin 12.
In de *Loop* wordt een teller steeds opgehoogd met Counter = Counter + 1 % Numpixels;.
Maar er wordt met % Numpixels Modulo gerekend met het aantal Neopixels. Dus zodra de teller daar overheen gaat wordt-ie weer 'gereset'.

```
#include <Adafruit_NeoPixel.h>

#define Pixelpin  12    // Digital IO pin connected to the NeoPixels. #define
Numpixels 24     // Er zijn 24 pixels.

Adafruit_NeoPixel Ledstrip = Adafruit_NeoPixel(Numpixels, Pixelpin, NEO_GRB + NEO_KHZ800);

void setup() {
Serial.begin(9600);
  Ledstrip.setBrightness(10); // 10 is een mooie waarde. 64 is veel te fel   Ledstrip.begin();
  Ledstrip.show(); // Initialize all pixels to 'off' }

int Counter = 0;

void loop() {
  Ledstrip.setPixelColor((Counter) % Numpixels, 0, 0, 0);
  Ledstrip.setPixelColor((Counter + 4) % Numpixels, 0, 0, 0);
  Ledstrip.setPixelColor((Counter + 8) % Numpixels, 0, 0, 0);
  Ledstrip.setPixelColor((Counter + 12) % Numpixels, 0, 0, 0);
  Ledstrip.setPixelColor((Counter + 16) % Numpixels, 0, 0, 0);
  Ledstrip.setPixelColor((Counter + 20) % Numpixels, 0, 0, 0);
  Counter = (Counter + 1) % Numpixels;
  Ledstrip.setPixelColor((Counter) % Numpixels, 255, 0, 0);
  Ledstrip.setPixelColor((Counter + 4) % Numpixels, 255, 0, 0);
  Ledstrip.setPixelColor((Counter + 8) % Numpixels, 255, 0, 0);
  Ledstrip.setPixelColor((Counter + 12) % Numpixels, 255, 0, 0);    Ledstrip.setPixelColor((Counter
+ 16) % Numpixels, 255, 0, 0);
  Ledstrip.setPixelColor((Counter + 20) % Numpixels, 255, 0, 0);
  Ledstrip.show();
delay(400);
}
```

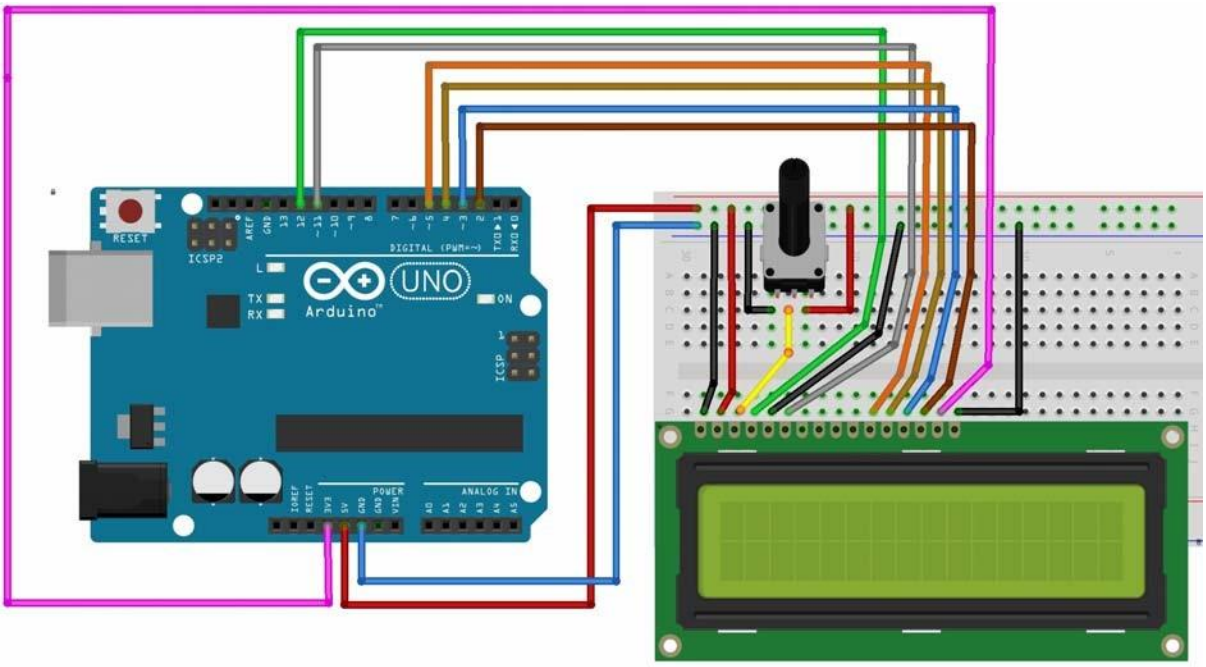
Neopixel ringen kunnen gekoppeld worden. Maar het blijft altijd wat het is.. Eén lange sliert ledjes. …

Project 23: Het LCD (Liquid Crystal Display)

In plaats van output naar de seriele monitor is het leuker en mooier om output naar een LCD display te sturen. De afkorting betekent Liquid Crystal Display. Ze zijn er in verschillende soorten. B&W (Blauw op wit), RGB, groot, klein…We gebruiken hier een veelvoorkomend type: het 16x2 B&W LCD display.
Het aansluiten is nog niet zo eenvoudig want er moeten in totaal 16 verbindingen gemaakt worden….

1. Druk het LCD display op het breadboard zodat er de *f* en *g* banen nog zichtbaar zijn en vanaf baan *h* alles bedekt wordt door het display. (Het 1e pinnetje komt links in de 3e rij terecht)
2. Verbind *RS* met pin 12, *E* met pin 11, *D4*, *D5*, *D6* en *D7* met pin 5, 4, 3, 2
3. Verbind *Vss* met Gnd en daarna *Vss* ook met de blauwe **– *lijn (Gnd)*** 4. Verbind *Vdd* met 5V en daarna de *Vdd* ook met de rode **+ *lijn***

5. Verbind *A* met de 3V pin en *K* met de – lijn. 6.
Verbind de *RW* met Gnd
7. Sluit een 10 k potmeter aan op + en - lijn en verbind het middelste contact met de *V0* aansluiting. Met de potmeter stel je (voor dit display) de hoeveelheid “backlight” in. Experimenteren!



Als voorbeeld worden in dit programma verschillende manieren gebruikt om iets op het display te zetten. In deze demo sketch worden verschillende opdrachten gebruikt.

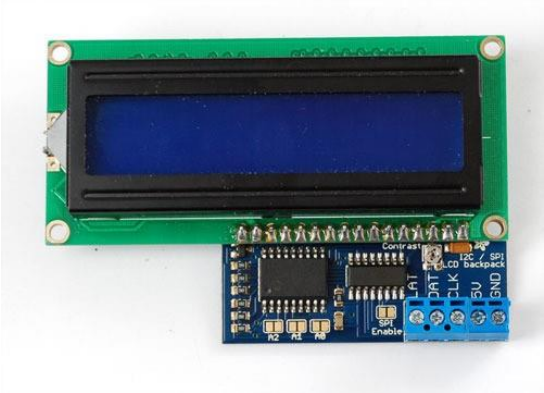
lcd.print("Hello, world!");	Print de tekst “Hello, world”
lcd.print(k);	Print het getal *k*
lcd.setCursor(0, 1);	Zet de cursor klaar om te schrijven op kolom 0, lijn1. (Lijn 0 is de eerste regel)
lcd.noDisplay();	Zet het display uit.
lcd.display();	Zet het display weer aan.
lcd.scrollDisplayLeft()	Elke keer dat dit wordt gebruikt springt de tekst één positie naar links.
lcd.scrollDisplayRight();	Elke keer dat dit wordt gebruikt springt de tekst één positie naar rechts. Door dit in een loopje te doen kun je een tekst als een lichtkrant laten lopen.

lcd.clear(); Maak het scherm leeg.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // initialize the library with the numbers of the interface pins

void setup() {   lcd.begin(16, 2);   // set up the LCD's number of
columns and rows:   lcd.print("Hello World!!");   // Print a
message to the LCD.
  lcd.clear();   lcd.print("Demo LCD Display!");   // Print a
message to the LCD. }

void loop() {
  for (int k = 0; k <= 10; k++) {   // Het printen van een veranderend
getal     lcd.clear();     lcd.setCursor(0, 0);     lcd.print("Een getal.....");
lcd.setCursor(0, 1);
    lcd.print(k);
delay(500);
  }
  delay(1000);
  lcd.clear();

  lcd.setCursor(0, 0);  // Het schrijven van een twee-regelige
tekst   lcd.print("Regel 1: ");   lcd.setCursor(0, 1);
lcd.print("Regel 2: ");
  delay(1000);
  lcd.clear();

  lcd.setCursor(0, 0);   lcd.print("Left and right scrolling");   delay(500); // Naar links scrollen
for (int k = 0; k <= 23; k++) {     lcd.scrollDisplayLeft(); //Elke keer dat dit wordt gebruikt
springt de tekst één positie naar links.
    //     lcd.scrollDisplayRight(); //Elke keer dat dit wordt gebruikt springt de tekst één positie naar rechts.
    delay(200);
  }
  delay(500); // Nu naar rechts scrollen   for (int k = 0; k <= 23; k++) {
lcd.scrollDisplayRight(); //Elke keer dat dit wordt gebruikt springt de tekst één positie naar links.
    delay(200);
  }
  delay(500);
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Graden
teken");
lcd.setCursor(0, 1);
lcd.print(char(223));
lcd.print("C");
delay(2000);
}
```

Project 24: LCD met I²C bus (Adafruit)

Het “klassieke” LCD display moet met veel draden aangesloten worden.
Maar zoveel aansluitdraden op een breadbord is onhandig, storingsgevoelig en het vermindert het aantal aansluitpinnen van de Uno. Een beter idee is gebruik te maken van de I2C bus op pin A4 en A5 van de Arduino.



I²C is een protocol dat al in 1979 door Philips ontwikkeld is. Veel Arduino componenten maken er gebruik van. Zie https://nl.wikipedia.org/wiki/I%C2%B2C-bus

Om te kunnen communiceren heeft I²C één master nodig en minimaal één slave. De master (de Arduino) heeft de controle over de I²C-bus en genereert het kloksignaal, startbit en stopbit. De slaves (display of andere I²C devices) communiceren alleen dan, nadat de master daartoe een verzoek stuurt.

Eén van de uitvoeringen is het Adafruit Backpack

In deze kit zit een 16x2 LCD display met een Adafruit Backpack expander. Dat regelt de I2C verbinding tussen de Uno en het display. Met een kleine schroevedraaier kan de helderheid ingesteld worden.

Aansluitingen:
5V met 5V, GND met Ground, DATA (SDA) met A4, CLocK (SCL) met A5 op de Uno) Tip:
Gebruik altijd een geLe draad voor de CLK lijn en een orAnje draad voor de DAT lijn. Dan werkt het ezelsbruggetje:
DATA=SDA=oRAnje=A’vieR’
Clock=CLK=geeL

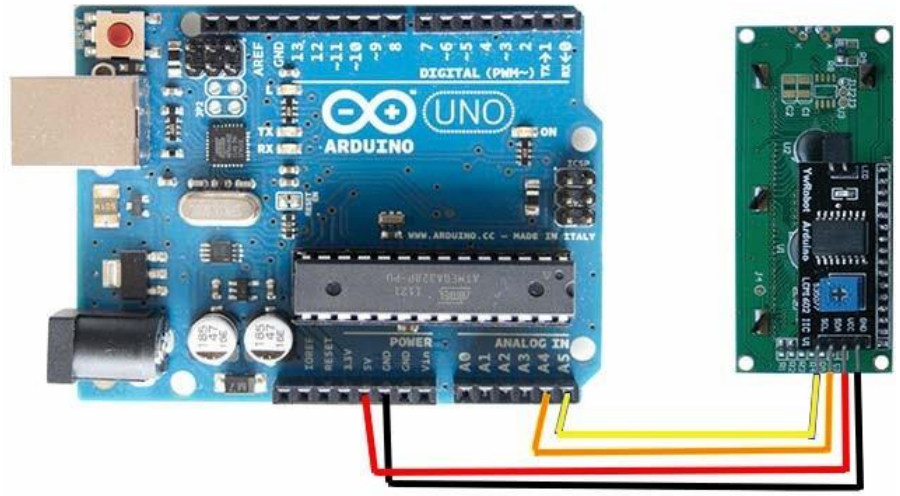
Zie ook https://learn.adafruit.com/i2c-spi-lcd-backpack

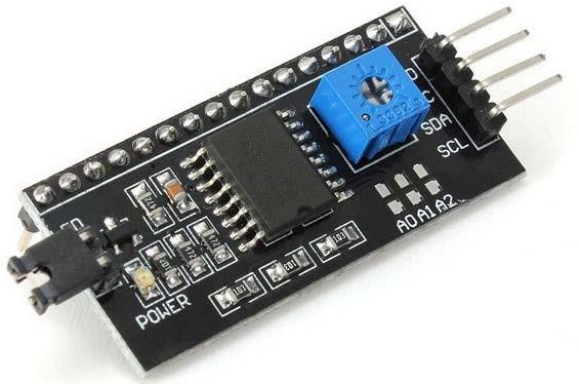
Hetzelfde programma als met het “klassieke” LCD scherm:

```
#include "Wire.h"                          // Nodig om de I2C bus te kunnen gebruiken #include
"Adafruit_LiquidCrystal.h"

Adafruit_LiquidCrystal lcd(0);

void setup() {   lcd.begin(16, 2);   lcd.begin(16, 2);   // set up the
LCD's number of columns and rows:   lcd.print("Hello World!!");   //
Print a message to the LCD.
  lcd.clear();
}

void loop() {
  // Het printen van een veranderend getal
for (int k = 0; k <= 10; k++) {
    lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Een getal.....");
lcd.setCursor(0, 1);
    lcd.print(k);
delay(500);
  }
  delay(1000);
lcd.clear();

  // Het schrijven van een twee-regelige tekst
lcd.setCursor(0, 0);   lcd.print("Regel 1: ");
lcd.setCursor(0, 1);   lcd.print("Regel 2: ");
  delay(1000);
lcd.clear();

  lcd.setCursor(0, 0);   lcd.print("Left and right scrolling");   delay(500); // Naar links scrollen   for
(int k = 0; k <= 23; k++) {     lcd.scrollDisplayLeft(); //Elke keer dat dit wordt gebruikt springt de
tekst één positie naar links.
    delay(200);
  }   delay(500); // Nu naar rechts scrollen   for (int k = 0; k <= 23; k++) {     lcd.scrollDisplayRight();
//Elke keer dat dit wordt gebruikt springt de tekst één positie naar links.     delay(200);
  }
  delay(500);   lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Graden teken");
lcd.setCursor(0, 1);
lcd.print(char(223));
lcd.print("C");
delay(2000);
}
```

Project 25: LCD met I²C bus (PCF8574)

Een andere mogelijkheid is gebruiken van de goedkope expander van https://www.hobbyelectronica.nl/ Dit (en dergelijke) shields zijn gebaseerd op de PCF8574 chip die voor de I²C communicatie zorgt.





Het Backpack Shield wordt meestal achterop het LCD scherm gemonteerd.
Het programma is vrijwel hetzelfde. Maar voor dit shield is de LiquidCrystal_I2C library nodig.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address
```

Elk I2C 'device' heeft een eigen adres. Op dit shield staan een paar kleine vlakjes met daarbij A0, A1 en A2. Door soldeerbruggen te maken kan een ander adres dan de default 0x27 worden ingesteld. (1 = Not Connected. 0 = Connected)

A0	A1	A2	HEX Address
1	1	1	0x27
0	1	1	0x26
1	0	1	0x25
0	0	1	0x24
1	1	0	0x23
0	1	0	0x22
1	0	0	0x21
0	0	0	0x20

Project 26: Stackable LCD met I²C bus (Adafruit met 5 buttons) basic

Meestal willen we ook wat schakelaars aansluiten.
Dat levert dan toch weer allerlei draden op.
Een interessante derde mogelijkheid is om een Adafruit I2C display met “aangebouwde” buttons te gebruiken:
https://www.adafruit.com/product/772
Dit is overigens geen ‘snel’ shield. Een lees en schrijf actie duurt ca. 90 msec. Dat betekent dat de buttons hooguit 10x per seconde kunnen worden uitgelezen. Dat is langzaam…..

Het mooie van dit shield is dat het –met enkele aanpassingen- stackable gemaakt kan worden.
Plaats het voorzichtig bovenop de Arduino, zó dat aan de rechterkent de pinnen precies aansluiten op de Arduino. Het resultaat: een I2C LCD, met 5 schakelaars, alles gaat via A4 en A5 én we kunnen nog steeds bij alle andere pinnen van de Arduino. (Het is wel handig om aan de zijkant stickers te plakken want je kunt niet goed meer zien welke pin nu waar zit….)
Met de potmeter op het shield kan de helderheid weer bijgeregeld worden. Een eenvoudig basisprogramma ziet er zó uit:

```
#include <Wire.h>
#include <Adafruit_MCP23017.h>
#include <Adafruit_RGBLCDShield.h>

Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();

void setup() {
lcd.begin(16, 2);
  lcd.print("Hello World!");
}

void loop() {
lcd.setCursor(0, 1);   int
time = millis();
lcd.print(time / 1000);
  int buttons = lcd.readButtons();

  if      (buttons)      {
lcd.clear();
lcd.setCursor(0, 0);
    if (buttons & BUTTON_UP)        lcd.print("UP ");     if
(buttons & BUTTON_DOWN) lcd.print("DOWN ");     if
(buttons & BUTTON_LEFT)    lcd.print("LEFT ");       if
(buttons & BUTTON_RIGHT) lcd.print("RIGHT ");     if
(buttons & BUTTON_SELECT) lcd.print("SELECT ");
  }
}
```

Hierin komen een paar interesante opdrachten voor.

- `int time = millis();` Berekent de tijd in milliseconde vanaf de start van het programma
- `int buttons = lcd.readButtons();` Leest de buttons uit.
- `if (buttons) {….{` Als buttons een waarde heeft, doe dan …….
- `if (buttons & BUTTON_UP) lcd.print("UP ");` Als buttons een waarde heeeft en het is BUTTON_UP, dan ....

Project 27: Een I²C scanner

Om met I²C devices te kunnen communiceren moet het adres bekend zijn. Meestal is dat 0x27 maar het kan ook een ander adres zijn. En als meerdere devices –de grote kracht van I²C!- aangesloten worden op de I2C bus mogen die adressen niet identiek zijn.

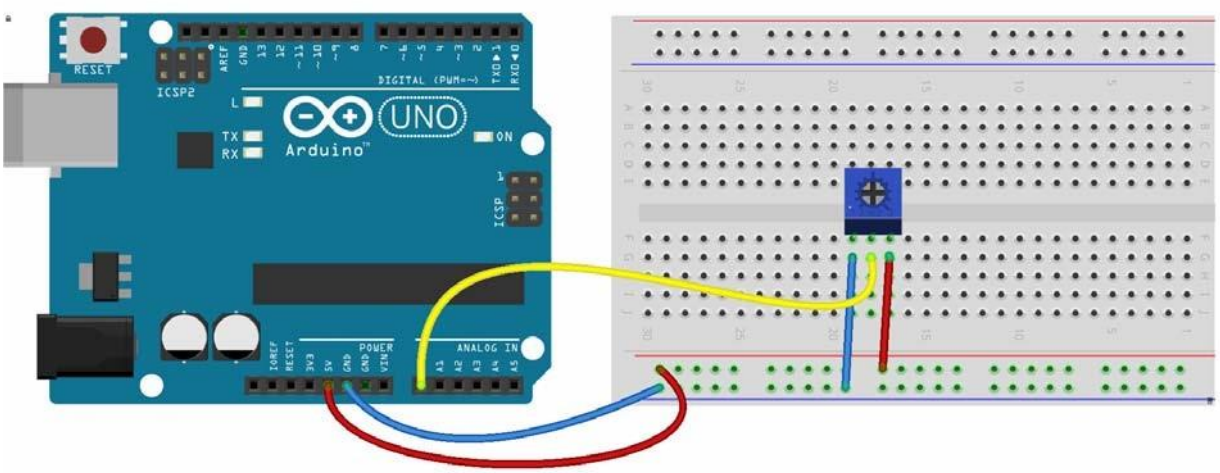
Het programma I²C scanner scant de I2C bus en geeft van de aangesloten devices het adres weer. Niets meer en ook niets minder! De werking ervan gaat hier veel te ver maar het is een handig programma.

```
#include <Wire.h>

void setup()
{
  Wire.begin();
  Serial.begin(9600);
  Serial.println("\nI2C Scanner");
}

void loop()
{
  byte error, address;
int nDevices;
  Serial.println("Scanning...");
nDevices = 0;
  for (address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of the Write.endTransmisstion to see if a device acknowledge the address.
Wire.beginTransmission(address);     error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
if (address < 16) Serial.print("0");
      Serial.print(address, HEX);
      Serial.println("  !");
nDevices++;
    }     else if (error
== 4)
    {
      Serial.print("Unknow error at address 0x");
      if (address < 16) Serial.print("0");
      Serial.println(address, HEX);
    }
  }
  if (nDevices == 0)Serial.println("No I2C devices found\n");   else
Serial.println("done\n");
  delay(5000);          // wait 5 seconds for next scan }
```

Project 28: Spanning meten met weergave op **LCD**

Vanaf dit moment gaat deze cursus er van uit dat er een I²C display is aangesloten op A4 en A5. In de voorbeeldprogramma's wordt uitgegaan van het display van project 25. Voor andere displays zal er wat aangepast moeten worden in de eerste regels van het programma.

Als eerste gaan we een spanning op een van de analoge ingangen meten. (Zie ook project 10)



Het programma is erg eenvoudig.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address

int SensorPin = A0;       // Input van de potmeter int
meetwaarde = 0;       // Een spanning, te meten op A0 float
echtewaarde = 0;    // float is een getal met decimalen

void setup()
{
  lcd.begin(16, 2);
lcd.setCursor(0, 0);
  lcd.print("Spanning in Volt");
}


void loop()
{   lcd.setCursor(0, 1);   meetwaarde = analogRead(SensorPin);
// lees de meetwaarde:
  echtewaarde = map(meetwaarde, 0, 1023, 0, 5000) / 1000.;    lcd.print(echtewaarde,
2);
  lcd.print("  Volt");
delay(600);
}
```

Project 29: Meten zonder delays!

Het vorige programma heeft één groot nadeel. In de “Loop” wordt de spanningsmeting uitgevoerd en daarna wordt het resultaat naar het LCD display geschreven. Om allemaal geknipper te voorkomen wordt er daarna 0.6 sec. gepauzeerd.
Maar tijdens zo’n delay gebeurd er even helemaal niets! Arduino staat doelloos te wachten.

Voor een simpele spanningsmeting is dat misschien nog acceptabel maar meestal moet het programma nog veel meer doen. Bijvoorbeeld controleren of er al ergens een button is ingedrukt.

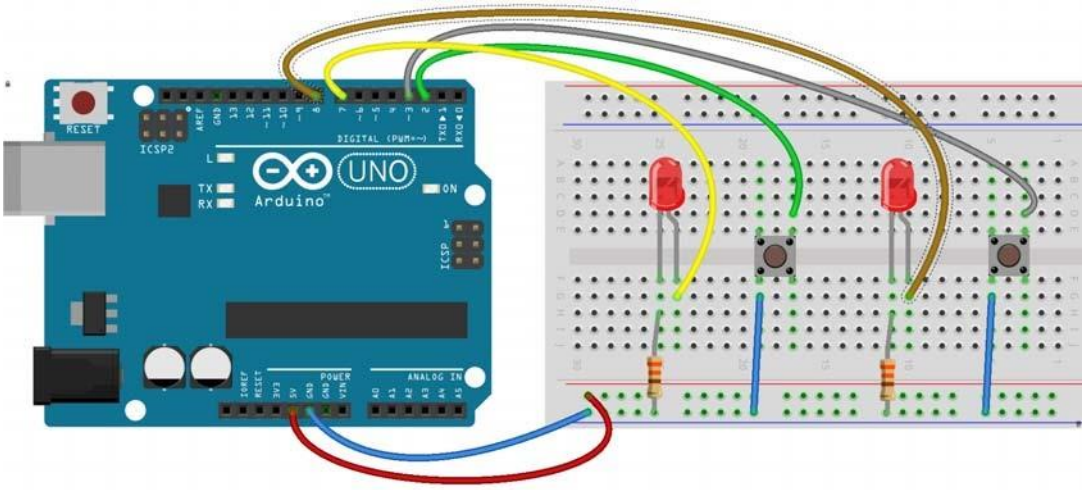
We zouden dus iets moeten hebben waardoor af en toe, (eenmaal per 600 mseconde) de spanning wordt gemeten terwijl het programma in de tussentijd blijft doorlopen.
Het komt er op neer dat we steeds op de klok kijken of het al tijd is om iets te doen.
Dit wordt “*polling*” genoemd.
In de volgende essentiele regels is te zien dat in de loop voortdurend een functie “Spannings_Meting” wordt uitgevoerd. Het lijkt alsof Arduino daar dan continue erg druk mee al zijn.
Maar… in die functie zelf wordt als allereerste kekeken of er al voldoende tijd verstreken is sinds de vorige keer. De opdracht millis() levert de tijd op vanaf het opstarten van de Arduino. Feitelijk dus een klok. De variabele time_to_Measure onthoudt de vorige tijd dat er een meting werd gedaan.
Àls er voldoende tijd is verstreken, n.l. Measure_Period dán wordt er een meting gedaan en op het display gezet. Natuurlijk moet dan met time_to_Measure = millis(); de tijd dat de meting het laatst werd uitgevoerd aangepast worden.

```
……. const int Measure_Period = 600;          // Update measurement every 600
msec. unsigned long time_to_Measure = 0; …..

void loop()
{
  Spannings_Meting();
}

void Spannings_Meting() {    if ((millis() - time_to_Measure) >
Measure_Period)  {      meetwaarde = analogRead(SensorPin);        //
lees de meetwaarde:
    echtewaarde = map(meetwaarde, 0, 1023, 0, 5000) / 1000.;
time_to_Measure = millis();      lcd.setCursor(0, 1);
lcd.print(echtewaarde, 2);
    lcd.print("  Volt");
  }
}
```

Deze manier van programmeren biedt veel voordelen. Het levert overzichtelijke code op. En het zorgt ervoor dat de Arduino ook nog andere dingen kan doen.

Project 30: Polling Led´s en buttons

Met deze *polling* techniek kunnen we nu iets doen dat eerder niet kon.. Twee led’s die onafhankelijk van elkaar in een ander tempo knipperen of flashen. De schakeling is als in project 4.



Het programma bestaat uit twee vrijwel identieke delen, voor button en led 1 en 2.

```
int Led1Pin               =  2;      // the number of the Led pin int
Led1State            = LOW;     // LedState used to set the Led
long Led1OnTime           = 150;     // milliseconds of on-time
long Led1OffTime          = 250;     // milliseconds of off-time
unsigned long Led1Lasttime = 0; int Led1Button            = 6;
enz….

void setup()
{
  pinMode(Led1Pin, OUTPUT);
pinMode(Led1Button, INPUT_PULLUP); enz.
}

void loop() {
  Led1(); enz….
}

void Led1() {
  if (digitalRead(Led1Button) == LOW && Led1State == HIGH && (millis() - Led1Lasttime) > Led1OnTime)  {
Led1State = LOW;  // Turn it off
    Led1Lasttime = millis();
    digitalWrite(Led1Pin, Led1State);  // Update the actual Led
  }
  if (digitalRead(Led1Button) == LOW && Led1State == LOW && (millis() - Led1Lasttime) > Led1OffTime)  {
Led1State = HIGH;  // Turn it on
    Led1Lasttime = millis();
    digitalWrite(Led1Pin, Led1State);  // Update the actual Led
  }
}
```

Project 31: Classes

Voor een paar led's kan het essentiele stukje code gewoon gekopieerd worden. Maar omdat nu voor 10 leds zo te doen is niet elegant. Dat kan eenvoudiger door gebruik te maken van een class Knipperled - Allereerst beschrijven we van die class welke variabelen in die *class* gebruikt worden.
- Daarna volgt in public de beschrijving van wat er bij de setup moet gebeuren.
- Daarna volgt in Update de beschrijving van wát "Knipperled" moet doen.

```
class Knipperled
  {
    int ledPin;      // the number of the LED pin
int ledState;
    long OnTime;     // milliseconds of on-time
long OffTime;    // milliseconds of off-time
unsigned long Lasttime;     int LedButton;

  public:
    Knipperled(int pin, long on, long off, int button)
     {
      ledPin = pin;
      LedButton           =           button;
pinMode(pin, OUTPUT);
      pinMode(button, INPUT_PULLUP);
      OnTime = on;
OffTime = off;       ledState
= LOW;
      Lasttime = 0;
    }

    void Update()     {
      if (digitalRead(LedButton) == LOW && ledState == HIGH && (millis() - Lasttime >= OnTime))
     {
       ledState = LOW;                // Turn it off
       Lasttime = millis();           // Remember the time
digitalWrite(ledPin, ledState);  // Update the actual LED
      }
      if (digitalRead(LedButton) == LOW && ledState == LOW && (millis() - Lasttime >= OffTime))
     {
       ledState = HIGH;               // Turn it on
       Lasttime = millis();           // Remember the time
digitalWrite(ledPin, ledState);  // Update the actual LED
      }
    }
};
```

En nu hebben we een object gedefinieerd dat we heel eenvoudig kunnen gebruiken.

```
Knipperled led1(2, 100, 400, 6);    // Geef led1 alle eigenschappen van het object Knipperled Knipperled
led2(3, 350, 350, 7);   // Geef ook led2 enz. enz. die eigenschappen

void setup()  { }  // Hier staat niets in!

void loop() {
  led1.Update();  // En voer die update functie uit op led1
led2.Update();  // Enz. voor led2, led3…… }
```

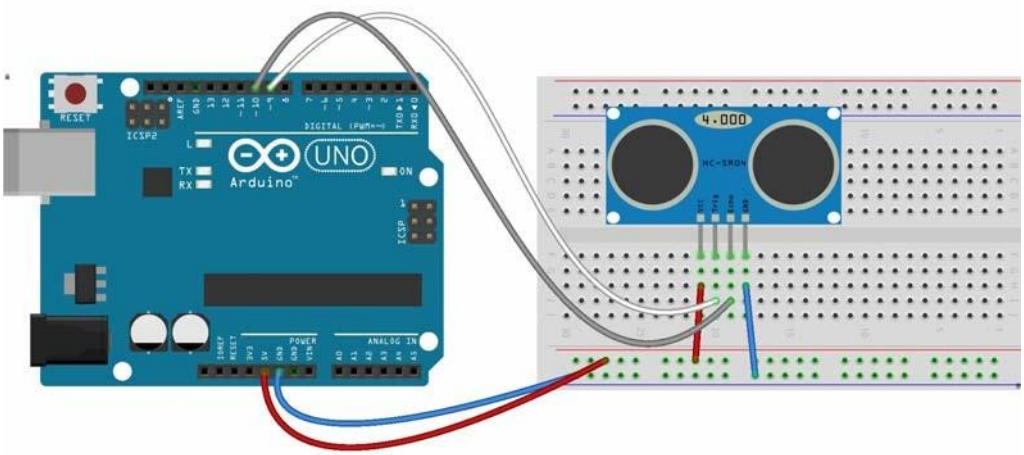
Project 32: Stackable LCD met I2C bus (Adafruit met 5 buttons) **polling**

Deze ‘polling’ techniek wordt vanaf nu steeds weer gebruikt. En dat betekent dat project 26 ook anders kan en moet. In de *loop* gebeurd niets anders dan Check_Buttons();
En daarin wordt alleen aandacht gegeven aan het display en de buttons als er 100 msec. verstreken is.

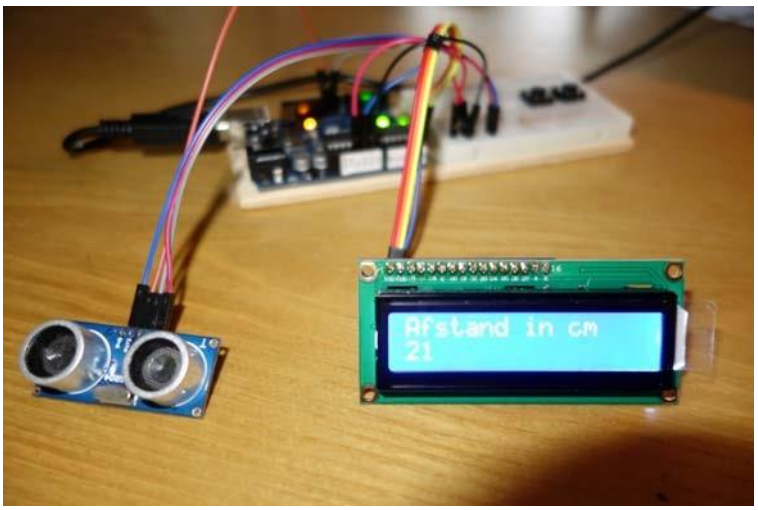
```
#include <Wire.h>          //I2C library
#include <Adafruit_MCP23017.h>      //Display library
#include <Adafruit_RGBLCDShield.h>    //Display library

Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield(); int button;
int lastbutton; int Read_Buttons_Period = 100;       // Check de buttons
elke 100 msec.
unsigned long time_to_Read_Buttons = 0;

void setup() {
lcd.begin(16, 2);
  lcd.setCursor(0, 0);          // set the cursor to column 0, line 1 (second row)   lcd.print("Regel
1");
  lcd.setCursor(0, 1);          // set the cursor to column 0, line 2 (second row)   lcd.print("Regel
2");
}

void loop() {
  Check_Buttons();          // Lees elke Read_Buttons_Period de buttons en onderneem actie. }

void Check_Buttons() {   if ((millis() - time_to_Read_Buttons) >
Read_Buttons_Period)  {     button = lcd.readButtons();        // Lees
button in.
    time_to_Read_Buttons = millis();
    //  Alléén iets doen als er een ándere button is ingedrukt dan daarvoor!!      if (lastbutton
!= button && (button & BUTTON_LEFT)) {  //Left button is ingedrukt      lcd.setCursor(0,
1);          // set the cursor to column 0, line 2 (second row)       lcd.print("Linker button   ");
    }
    if (lastbutton != button && (button & BUTTON_RIGHT)) {     //Rechter button is ingedrukt
lcd.setCursor(0,  1);                              // set the cursor to column 0, line 2 (second row)
lcd.print("Rechter button  ");
    }
    if (lastbutton != button && (button & BUTTON_UP))       {   //Up button is ingedrukt
lcd.setCursor(0, 1);               // set the cursor to column 0, line 2 (second row)       lcd.print("Up
button      ");
    }
    if (lastbutton != button && (button & BUTTON_DOWN))    {  //Up button is ingedrukt
lcd.setCursor(0, 1);                  // set the cursor to column 0, line 2 (second row)
lcd.print("Down button    ");
    }
    if (lastbutton != button && (button & BUTTON_SELECT))    {  //Up button is ingedrukt
lcd.setCursor(0, 1);                    // set the cursor to column 0, line 2 (second row)
lcd.print("Selectie button ");
    }   }   lastbutton = button; // Dit is
belangrijk!!
}
```

Project 33: Ultrasone afstandsmeting

Een ultrasoon sensor meet afstand door een geluidssignaal, een "*Ping"* uit te zenden.
Die "*Ping*" kaatst terug en wordt ontvangen door de sensor. De tijd tussen zenden en ontvangen wordt gemeten. Geluid gaat 340 meter per seconde en legt daarom in 1 microseconde 1/29 centimeter af. We moeten de tijd die het duurt tussen zenden en ontvangen delen door 29 en dan nog eens door 2 want het geluid moet heen en terug. Dán hebben we de afstand in centimeter.



De ultrasoon zender/ontvanger HC-SR04 heeft 4 pinnen: VCC aan +5V, GND aan GND, TRIG aan pin 9 (bijvoorbeeld), ECHO aan pin 10 (bijvoorbeeld). Aansluiten kan via het breadboard maar ook met 4 malefemale jumpers. Gebruik het breadboard voor de Vcc en Gnd aansluitng van LCD en sensor. Verder wordt het I2C display aangesloten op A4 en A5.



Het programma werkt weer met polling van de sensor, hier om de 500 msec.

```
#include <NewPing.h>                // Library voor de Ultrasoon sensor
#define TRIGGER_PIN  9          // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     10         // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 200  // Maximum distance in cm. Maximum distance is 400-500 cm.
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); int afstand = 0;

const int Measure_Period = 500;         // Update measurement every 500 msec.
unsigned long time_to_Measure = 0;

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address

void setup() {
lcd.begin(16, 2);
lcd.print("Ultrasoon");
delay(2000);
lcd.setCursor(0, 0);
  lcd.print("Afstand in cm   ");
}

void loop() {
  Ultrasoon_Meting();
}

void Ultrasoon_Meting() {   if ((millis() - time_to_Measure) > Measure_Period)  {
afstand = sonar.ping() / 58; // Send ping, get ping time, convert microseconds to cm.
time_to_Measure = millis();
    lcd.setCursor(0, 1);
lcd.print("               ");
lcd.setCursor(0, 1);
lcd.print(afstand);
  }
}
```

Project 34: Ultrasoon parkeeralarm

Een parkeeralarm in een auto werkt ook met ultrasone afstandsmeting. Als de afstand kleiner wordt dan een bepaalde waarde gaat een piepje klinken. Dat lijkt eenvoudig. Bijvoorbeeld iets als:

```
void Ultrasoon_Meting() {   if ((millis() - time_to_Measure) > Measure_Period)  {
afstand = sonar.ping() / 58; // Send ping, get ping time, convert microseconds to cm.
time_to_Measure = millis();     lcd.setCursor(0, 1);     lcd.print("              ");
lcd.setCursor(0, 1);     lcd.print(afstand);
   if(afstand<5) {Tone(buzzerpin,440);}
   if(afstand>=5 && afstand<15) {Tone(buzzerpin,220);}
if(afstand>=15) {noTone(buzzerpin);}
  }
}
```

Maar het combineren van ultrasoon met audio op deze manier gaat fout omdat Arduino voor (buzzerpin,440) van een *timer* gebruik maakt, een ingebouwde stopwatch. En Arduino blijkt dezelfde timer te gebruiken voor de Ultrasoon sensor als voor de beeper!
Het gaat allemaal wel goed als je de NewTone library gebruikt en niet het commando Tone(buzzerpin, frequentie, tijdsduur) gebruikt maar dat vervangt door NewTone(buzzerpin, frequentie, tijdsduur)

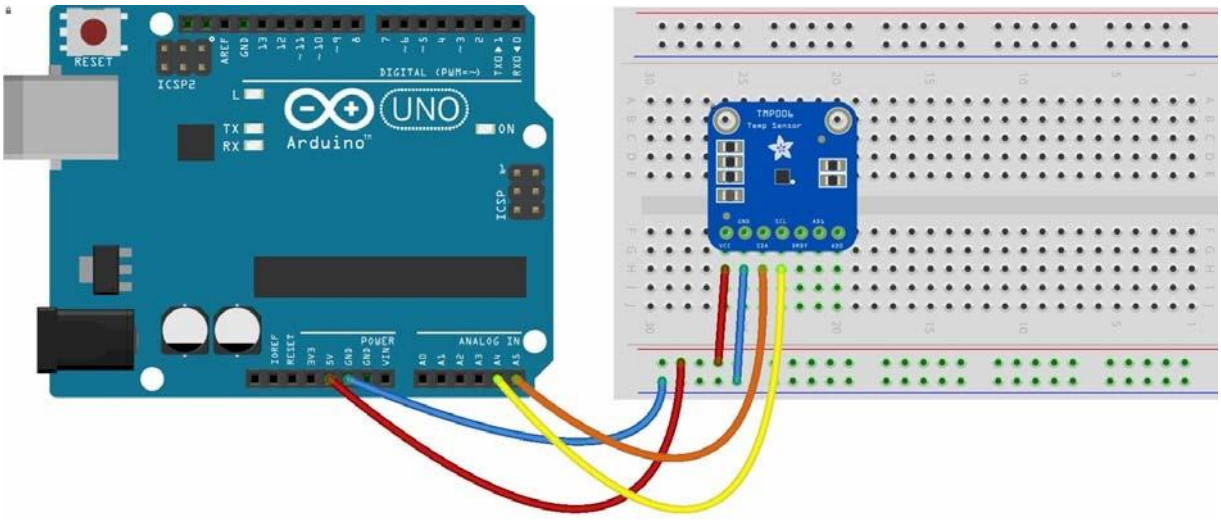
Een tweede probleem is dat we geen delays kunnen gebruiken. Tijdens een delay ligt de Arduino feitelijk "plat". Voor een Parkeeralarm niet zo heel handig…..
Dit is een probleem dat niet eenvoudig is om op te lossen. Een complete melodie *'Non-blocking'* afspelen als de afstand onder een grenswaarde komt vraagt om een apart audio shield dat de afhandeling van sounds verzorgd en daarmee de Arduino ontlast.
Maar alleen een piepertje laten klinken gaat prima met de *Polling* techniek die bij het project "Polling Led´s en buttons" is gebruikt.

Een volgend probleem is dat we die afstand eigenlijk voortdurend willen meten, b.v. elke 100 msec. maar dat het display dat niet elke 100 msec. hoeft weer te geven. Dus ook voor de weergave op het display is die *polling* techniek nodig. Het mooie is dat de *loop* nog steeds mooi kort en duidelijk is.

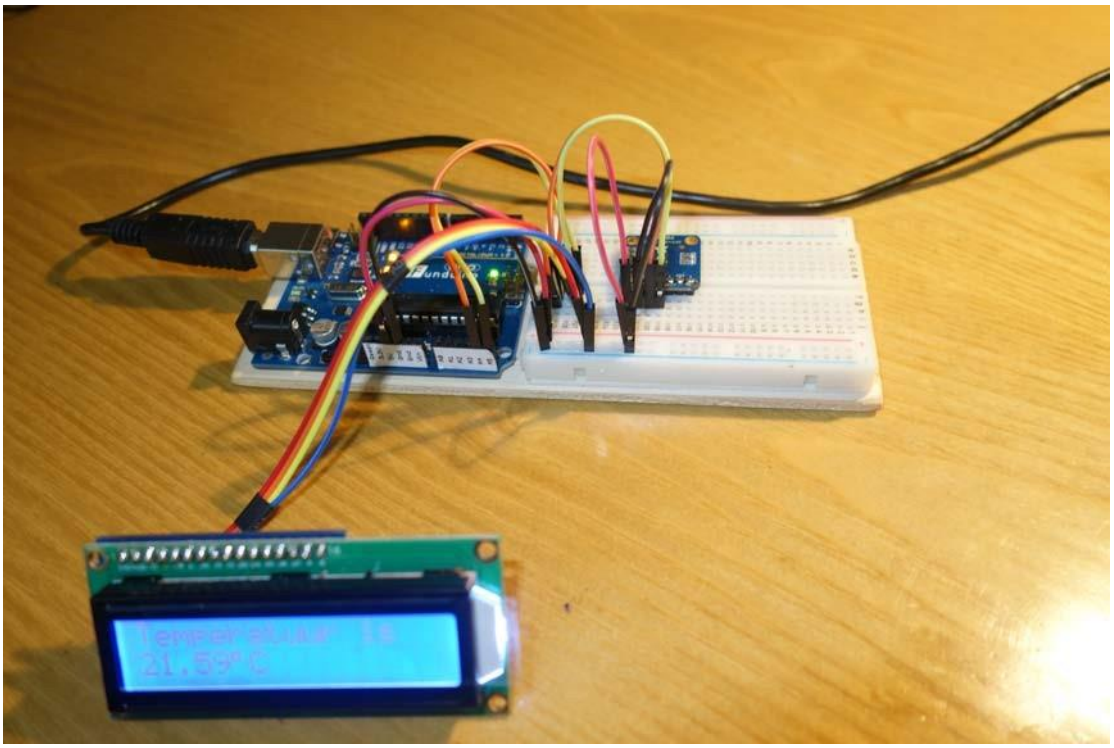
```
// Declaraties voor de ultrasoon sensor #include
<NewPing.h>
#define TRIGGER_PIN  9                  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     10                 // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 100          // Maximum distance in cm. Maximum distance is 400-500 cm.
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
const int Measure_Period = 100;          // Update measurement every 100 msec.
unsigned long time_to_Measure = 0; int afstand = 0;

// Declaraties voor de beeper
#include <NewTone.h>
int BeeperPin               =  2;     // the number of the Beeper pin int
BeeperState           = LOW;     // BeeperState used to set the Beeper
long BeeperOnTime           = 150;     // milliseconds of on-time long
BeeperOffTime          = 250;     // milliseconds of off-time unsigned long
BeeperLasttime = 0;
```

```
// Declaraties voor het I2C display
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address
const int Display_Period = 500; // Update display every 500 msec unsigned long
time_to_Display = 0;

void setup() {
lcd.begin(16, 2);
lcd.print("Ultrasoon");
delay(2000);
lcd.setCursor(0, 0);
  lcd.print("Afstand in cm   ");
}

void loop() {
  Ultrasoon_Meting(); // Every 100 msec
  Update_LCD_Display(); // Every 500 msec
  Beeper();
}

void Ultrasoon_Meting() {   if ((millis() - time_to_Measure) > Measure_Period)  {     afstand =
sonar.ping() / 29 / 2; // Send ping, get ping time, convert microseconds (uS) to cm.
time_to_Measure = millis();
  }
}

void Beeper() {
  if (BeeperState == HIGH && (millis() - BeeperLasttime) > BeeperOnTime)  {
    BeeperState = LOW;  // Turn it off
    BeeperLasttime = millis();
    noNewTone(BeeperPin);
  }
  if (afstand < 15 && afstand > 0 && BeeperState == LOW && (millis() - BeeperLasttime) > BeeperOffTime)  {
BeeperState = HIGH;  // Turn it on
    BeeperLasttime = millis();
    NewTone(BeeperPin, 440);
  }
}

void Update_LCD_Display() {
  if ((millis() - time_to_Display) > Display_Period) {
lcd.setCursor(0, 0);     lcd.print("Afstand in cm");
    if (afstand > 0) {
lcd.setCursor(0, 1);       lcd.print("
");       lcd.setCursor(0, 1);
lcd.print(afstand);
      time_to_Display = millis();
    }
  }
}
```

Project 35: Contactloos temperatuur meten met een InfraRood detector



Temperatuur kunnen we meten door het infrarode licht te meten dat wordt uitgestraald door een warm voorwerp. Het mooie is dat daarvoor geen direct contact nodig is. Een voorbeeld van zo'n contactloze temperatuurmeter is de TMP006. Deze sensor meet de hoeveelheid infrarood straling die er op valt.
Let op! Deze sensor is erg kwetsbaar! Niet aanraken! (Het is een kontaktloze temperatuursensor…..) Ook deze sensor werkt met een I2C bus. De combinatie met het LCD display is nu niet moeilijk meer. We hebben nu 2 'devices' die aan de I2C bus 'hangen'. Zoalng ze niet hetzelfde adres hebben is dat geen probleem. De opbouw lijkt nu wat rommeliger maar is nog steeds logisch.

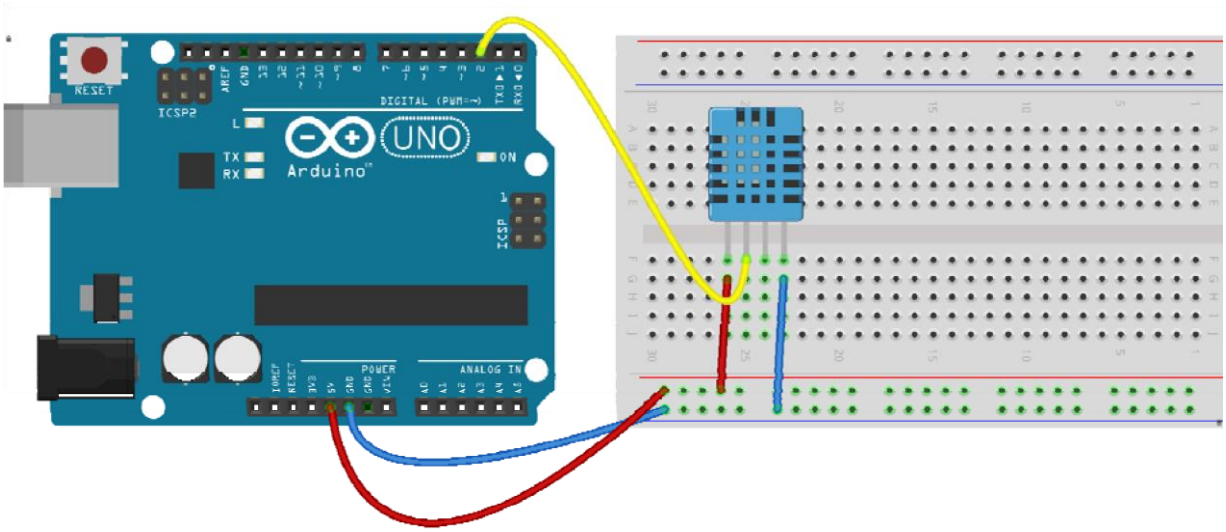


Én op de seriële monitor én op het LCD display wordt het graden teken achter de temperatuur gezet.
Bij het lcd display door lcd.print(char(223));
Op de Seriële monitor met Serial.print(char(194)); Serial.print(char(176));

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_TMP006.h"
#include <LiquidCrystal_I2C.h>

// SCL connects to the I2C clock pin. On the Uno, this is A5. //
SDA connects to the I2C data pin. On the Uno, this is A4.

Adafruit_TMP006 Sensor006; //Adafruit_TMP006 tmp006(0x41);  // start with a different i2c address!

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address

const int Measure_Period = 2000;          // Update measurement every 2000 msec.
unsigned long time_to_Measure = 0;

void setup() {
Serial.begin(9600);   lcd.begin(16,
2);
  lcd.print("Temperatuur is ");
  lcd.setCursor(0, 0);
  Sensor006.begin(TMP006_CFG_16SAMPLE);
}

void loop() {
  Temperatuur_Meting();
}

void Temperatuur_Meting() {   if ((millis() -
time_to_Measure) > Measure_Period)  {     float
object_temp = Sensor006.readObjTempC();
    time_to_Measure = millis();
Serial.print("De temperatuur is ");
    Serial.print(object_temp);
    Serial.print(char(194)); // Voor het graden-tekentje op de Seriele Monitor
Serial.print(char(176));     Serial.println("C");     lcd.setCursor(0, 1);
lcd.print(object_temp);
    lcd.print(char(223)); // Voor het graden-tekentje op de LCD
lcd.print("C");
  }
}
```

Project 36: Vochtigheid, temperatuur en dauwpunt met een DHT11

Er bestaan sensoren die vochtigheid en temperatuur meten. Een voorbeeld is de DHT11. In deze sensor zit een weerstand die reageert op vochtigheid, een NTC weerstand en een complete microprocessor. Die zorgt er voor dat alle meetgegevens over 1 pin worden verzonden.
Deze sensor werkt niet via I2C maar via een ander protocol. Nu is het versturen over één verbinding per definitie langzamer dan informatie versturen over meerdere verbindingen. Maar voor een sensor die trage veranderingen registreert zoals luchtvochtigheid is dit natuurlijk geen probleem. Uit de relatieve *vochtigheid* en de *temperatuur* wordt ook het *dauwpunt* berekend.



Opnieuw gebruiken we de techniek van het *pollen* delays te vermijden.
Maar, we willen achtereenvolgens de vochtigheid, de temperatuur en het dauwpunt op het display weergeven, zónder delay's te gebruiken.

In void DHT11_Meting() is daarvoor een slimmigheid ingebouwd. Een variabele meetwaarde loopt door de opdracht meetwaarde = (meetwaarde + 1) % 3 van 0 tot 2. Afhankelijk van die waarde wordt óf de vochtigheid, óf de temperatuur óf en het dauwpunt op het display weergegeven. Geen delays om de ene na de andere op het display te zetten. Delays are evil!

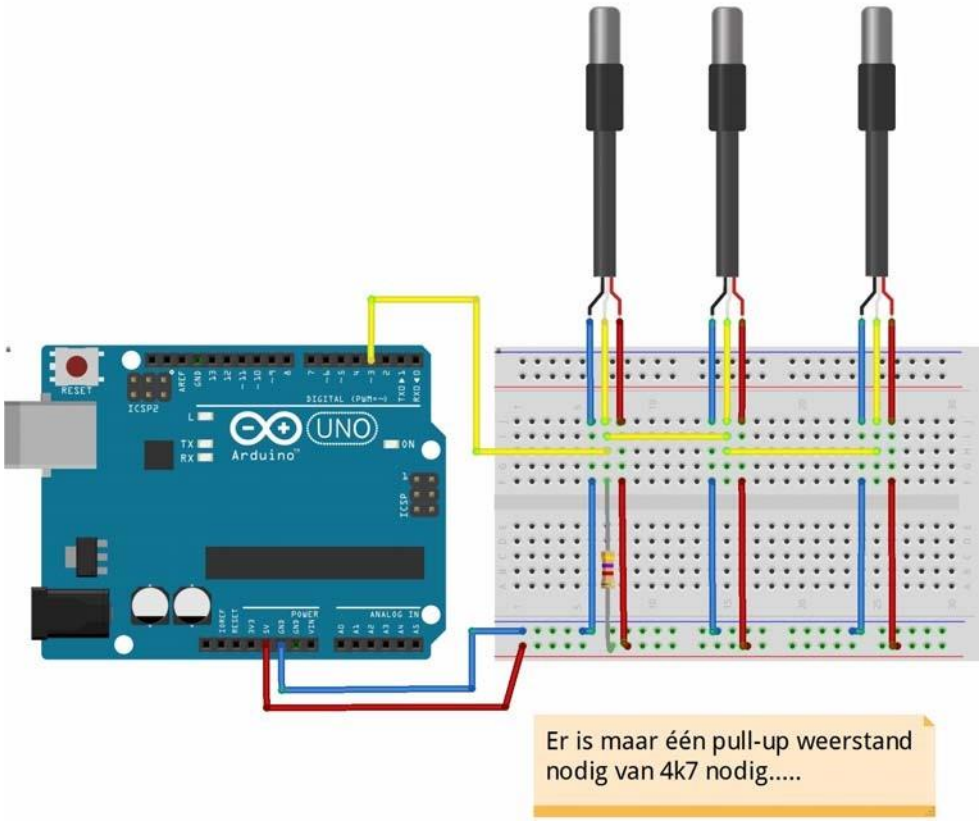
```
#include <Wire.h>
#include <dht11.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address dht11
DHT11sensor;

const int Measure_Period = 2000;          // Update measurement every 2000 msec.
unsigned long time_to_Measure = 0; int meetwaarde = 2;

void setup() {
Serial.begin(9600);   lcd.begin(16,
2);
  lcd.print("DHT11 sensor: ");
  lcd.setCursor(0, 0);
  DHT11sensor.attach(2); // DHT11 met -, + en data
}

void loop() {
  DHT11_Meting();
}

void DHT11_Meting() {
  if ((millis() - time_to_Measure) > Measure_Period)  {
meetwaarde = (meetwaarde + 1) % 3;
    if (meetwaarde == 0)  {       int
chk = DHT11sensor.read();
lcd.clear();       lcd.setCursor(0, 0);
lcd.print("Humidity:       ");
lcd.setCursor(0, 1);
     lcd.print(DHT11sensor.humidity, 1);
lcd.print(" %");
    }
    if (meetwaarde == 1)  {
lcd.clear();       lcd.setCursor(0,
0);       lcd.print("Temperatuur:
");       lcd.setCursor(0, 1);
     lcd.print(DHT11sensor.temperature, 1);
lcd.print(" ");       lcd.print(char(223));
lcd.print("C");
    }
    if (meetwaarde == 2)  {       lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Dauwpunt:       ");
lcd.setCursor(0, 1);
lcd.print(DHT11sensor.dewPoint(), 2);
lcd.print(" ");       lcd.print(char(223));
lcd.print("C");
    }
    time_to_Measure = millis();
  }
}
```

Project 37: Een waterdichte thermometer met een DS18B20

Er bestaan waterdichte temperatuursensoren.
Eén ervan is de DS18B20 die temperaturen kan meten van -55°C to +125°C

Een DS18B20 werkt met een speciaal “protocol”, het ‘*OneWire*’ protocol.
Vaak wil je niet één maar meerdere temperaturen tegelijkertijd op verschillende plekken meten. Als je heel veel sensoren hebt, heb je ook heel veel draden. Het OneWire protocol maakt het mogelijk om alle sensoren achter elkaar aan te plaatsen. Je kunt maximaal 127 sensoren achter elkaar in een meting opnemen. Er zijn daarbij twee mogelijkheden.

a. Elke sensor krijgt -, + en een “data” aansluiting. (Zoals in de tekening hieronder)
b. Elke sensor krijgt zijn voeding uit de “data lijn”. De – is meestal toch al aanwezig, b.v. het ijzeren frame van de machine (of de auto). Dan heb je alleen nog maar die ene “data” lijn. Dat is het echte OneWire protocol.

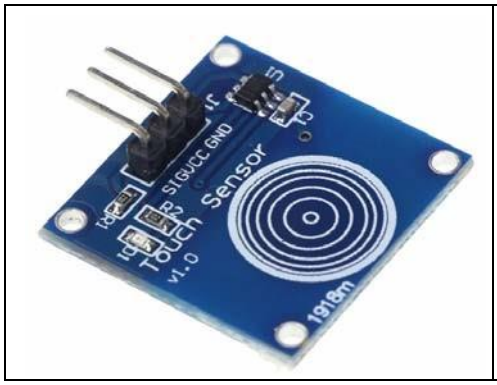


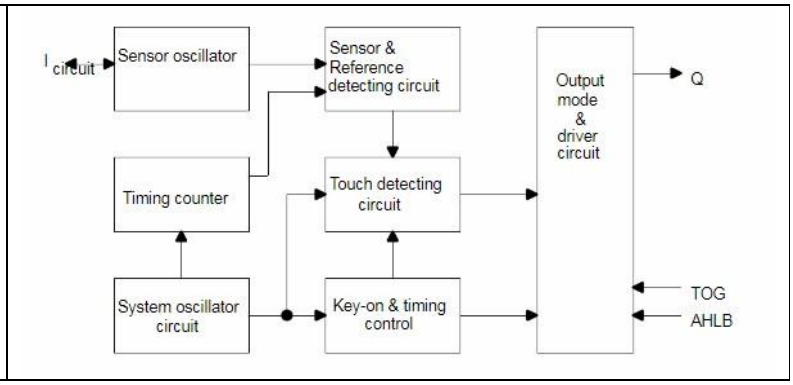
Hoe werkt dit nu? Dit zijn "*slimme*" sensoren. In de temperatuursensor zit behalve een sensor ook een complete microprocessor. Arduino vraagt welke sensoren er aanwezig zijn. Ze melden zich met een 'adres'. Daarna kan Arduino vragen naar de waarde van de temperatuursensor op dat 'adres'.
Als we de Onewire bibliotheek en de DallasTemperature bibliotheek geïnstalleerd hebben hoeven we ons over de details geen zorgen meer te maken.
De hiervolgende sketch is voor één temperatuursensor met weergave op I2C LCD. Voor de uitbreiding naar meerdere sensoren zie de voorbeelden bij de library.

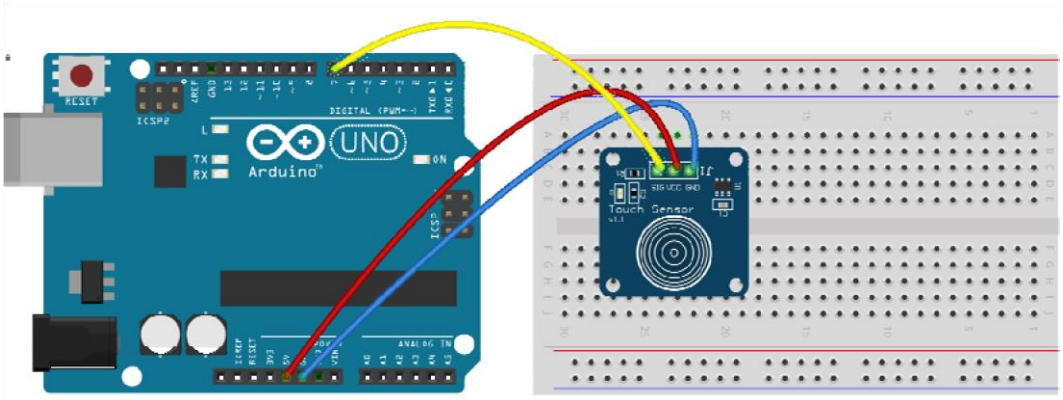
```
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 2 // De temperatuursensor is met een 4K7 Pull up weerstand op pin 2 aangesloten.  OneWire
oneWire(ONE_WIRE_BUS);    // Setup a oneWire instance to communicate with any OneWire devices
DallasTemperature sensors(&oneWire);   // Pass our oneWire reference to Dallas Temperature.
DeviceAddress Sensor;                          // arrays om de adressen van meerdere sensoren in op te slaan
// DeviceAddress Sensor1,Sensor2,Sensor3;    // arrays om de adressen van meerdere sensoren in op te slaan

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address const
int Measure_Period = 1000;          // Update measurement every 2000 msec.
unsigned long time_to_Measure = 0;

void setup()
{
  Serial.begin(9600);   // start serial port   lcd.begin(16,
2);
  lcd.print("DS18B20 sensor  ");
  lcd.setCursor(0, 0);
  sensors.begin();// Zoek sensoren op de bus   sensors.getAddress(Sensor,
0);
  sensors.setResolution(Sensor, 12);  // set the resolution to 9 bit   delay(1000);
}

void loop()
{
  Temperatuur_Meting();
}

void Temperatuur_Meting() {
  if ((millis() - time_to_Measure) > Measure_Period)  {
    sensors.requestTemperatures(); // Send the command to get temperatures
lcd.clear();     lcd.setCursor(0, 0);     lcd.print("Temperatuur:    ");
lcd.setCursor(0, 1);
    lcd.print(sensors.getTempC(Sensor), 2);
lcd.print(" ");     lcd.print(char(223));
lcd.print("C");
    time_to_Measure = millis();
  }
}
```

Project 38: Een On/Off led met Touch sensor (aanraakschakelaar)

Aanraakschakelaars zijn meestal capacitieve aanraakschakelaars.
Je kunt ze met losse electronica onderdelen maken maar zoals meestal heeft iemand alles wat nodig is al in een klein chipje gestopt. Hier het schematische opbouw van de TT223B en de schematische opbouw. Het werkt dus doordat de aanraking een oscillatorsignaal iets veranderd. Om te voorkomen dat een aanraking meerdere aan-uit's genereert is er nog wat omheen geplaatst om te "*ontdenderen*".







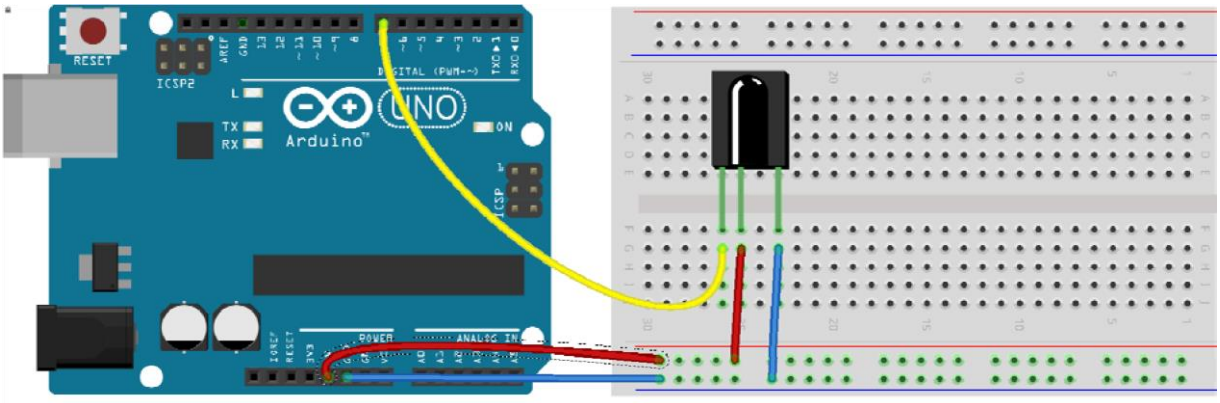
De volgende schets zorgt ervoor de de Led op de Arduino wordt 'getoggled'.
Er volgt actie als de schakelaar wordt geactiveerd. De essentiele regels uit de sketch:

```
int TouchPin = 7; // Pin for capacitive touch sensor int
led = 13;
boolean currentState = LOW; boolean
lastState = LOW;
boolean LedState = LOW;


void loop() {
  currentState = digitalRead(TouchPin);
  if (currentState == HIGH && lastState == LOW) {
delay(5);
    if (LedState == HIGH)  LedState = LOW;
else LedState = HIGH;
  }
  lastState = currentState;
digitalWrite(led, LedState); }
```

Project 39: Een Infrarood afstandsbediening met ‘Keyes’ afstandsbediening

Infrarood afstandsbediening is redelijk gestandaardiseerd. Maar het verschilt wel een beetje per afstandsbediening dus de volgende sketch bevat een lijstje codes.

De volgende sketch ontvangt IR commando’s en zet een tekst op de Seriële monitor.
Sluit de TSOP4838 IR fotodiode als volgt aan:
Vanaf de bolle ontvangstkant gezien van links naar rechts Data Vcc Gnd! Let op: dat is anders dan de specificaties op internet! Bij Vcc en Gnd andersom wordt deze fotodiode heel heet!!



Omdat de gebruiker op elk willekeurig moment op een knop kan drukken is nu die *polling* techniek niet zo goed bruikbaar. Arduino moet continue kijken of er iets binnenkomt op de IR poort.

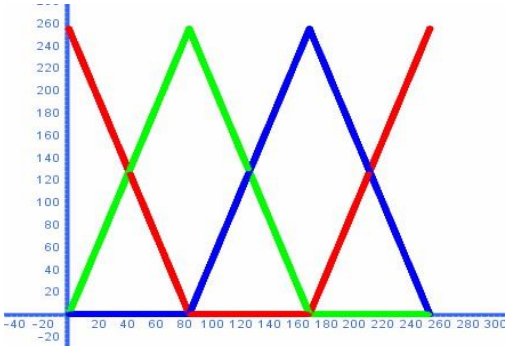
```
#include <IRremote.h>

const int RECV_PIN = 7; // the pin where you connect the output pin of TSOP4838

#define KEYES_Up 25245
#define KEYES_Left 0xFF22DD
#define KEYES_Ok 0xFF02FD enz…

unsigned int value;
IRrecv irrecv(RECV_PIN);
decode_results results;

void setup()
{
  Serial.begin(9600);   // you can comment this line
  Serial.println("Start IR ontvangst");   irrecv.enableIRIn();
// Start the receiver
}

void loop() {
  if (irrecv.decode(&results)) {
unsigned int value = results.value;
switch (value) {       case
(KEYES_Up):
       Serial.println("KEYESRemote Up");
break;       case KEYES_Left:
       Serial.println("KEYESRemote Left");
break;       case KEYES_Ok:
       Serial.println("KEYESRemote OK");
break;       case KEYES_Right:
       Serial.println("KEYESRemote Right");
break;       case KEYES_Down:
       Serial.println("KEYESRemote Down");
break;       case KEYES_1:
       Serial.println("KEYESRemote 1");
break; enz…

     case KEYES_Hekje:
       Serial.println("KEYESRemote #");
break;
    }
   irrecv.resume(); // Receive the next value
}
}
```

Project 40: Een fading RGB -Led

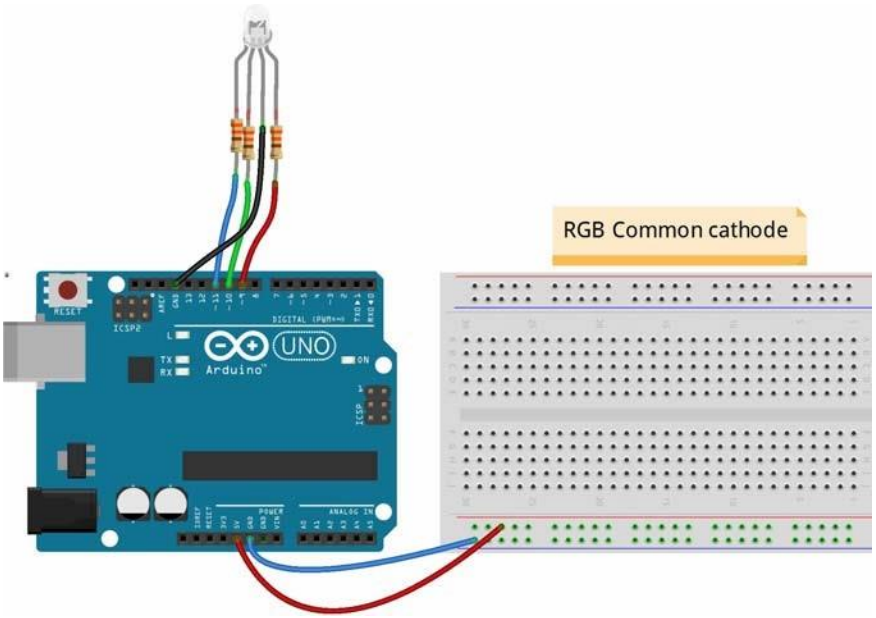
LED's branden wel of ze branden niet. Maar door ze heel snel aan en uit te zetten lijken ze gedimd. Dat snel aan/uit schakelen gaat het makkelijkste met de speciale Digitaloutputs pen 3,5,6,9,10 en 11. Met deze pennen is PulseWidthModulation mogelijk. Met een analogWrite commando naar deze pinnen wisselt daarna de toestand van zo'n pin van hoog naar laag in een bepaald tempo.
Bij analogWrite(255) staat de pin voortdurend aan, bij analogWrite(127) maar voor de helft. Een Led die is aangesloten op deze pinnen kun je dus dimmen!

Het wordt nog leuker als we een RGB led gebruiken. Dat zijn eigenlijk 3 Led's in één huisje. Een Rode led, een Groene led en een Blauwe led. Door van elke led te intensiteit te variëren kun je alle kleuren van de regenboog maken. RGB Led's zijn er in 2 soorten. Gemeenschappelijke + (Anode) of Gemeenschappelijke – (Kathode). Zoals altijd moeten er weerstanden van 330 Ohm gebruikt worden om de stroom te begrenzen.
Eigenlijk net zo makkelijk om die weerstanden vlak bij de Led te solderen…… De aansluitdraden hebben een vrij logische kleur. Zwart = Gnd, en de rest is RGB.

De code is vrij duidelijk. In de *Loop* loopt een ColorValue van 0 tot 255 en terug en deze wordt vertaald –op een wat onduidelijke manier- naar de juiste R,G en B waarden die naar de LED's gestuurd worden. Het komt er op neer dat als ColorValue loopt van 0-255 dat R, G en B zich gedragen zoals in bijgaande figuur.



Omdat de Led's in intensiteit aangestuurd moeten worden zijn alleen de PWM pinnen 3, 5, 6, 9, 10 en 11 bruikbaar.
Die Led aansluitingen direct op de Arduino 'prikken' levert slechte kontakten op. Handiger is om de vier aansluitingen van een '*pinheader'* te voorzien. Dat geeft een veel betrouwbaarder verbinding dan losse pinnen.
Mooi naast elkaar liggende PWM pinnen zijn 9,10 en 11.
Zwart moet aan Gnd. Gelukkig zit er aan die kant van de Arduino ook een Gnd aansluiting.



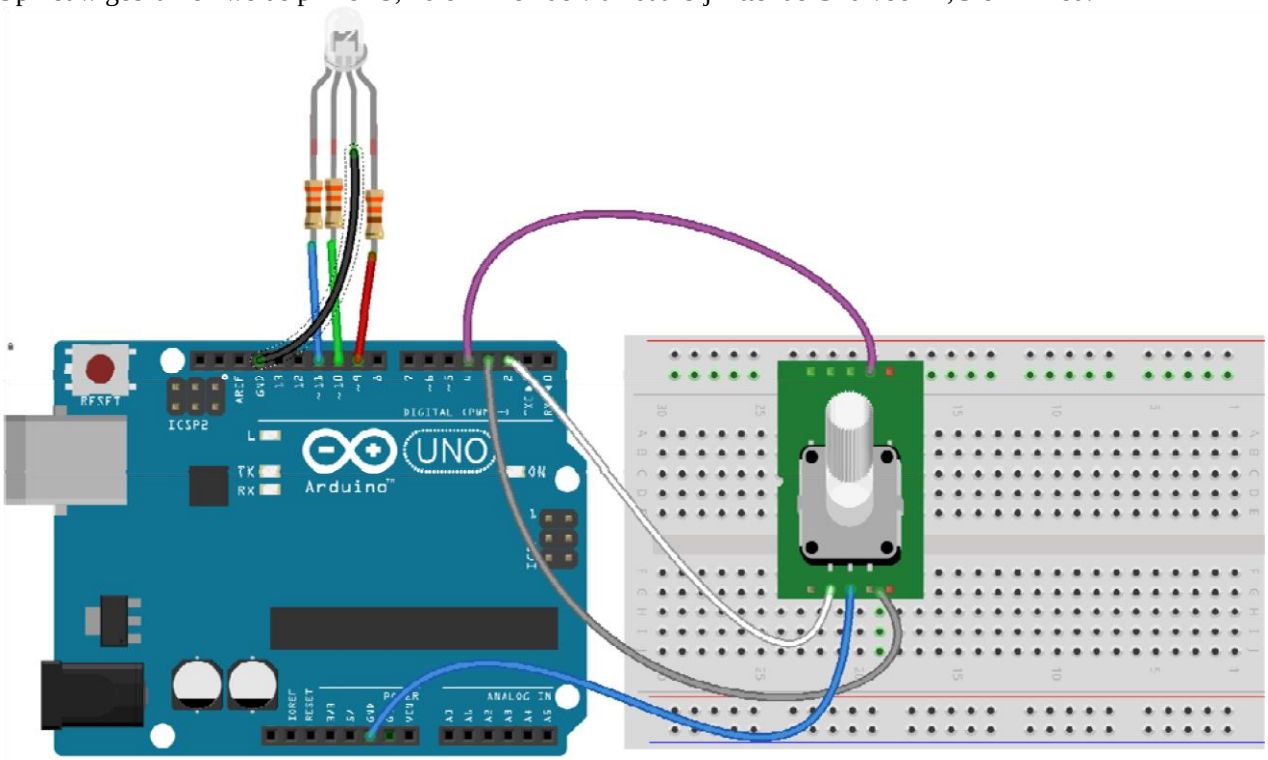
```
// RGB Auto Fader. Loopt door het spectrum van kleuren heen in 255 stappen.

int RedLed   = 9; int
GreenLed = 10;
int BlueLed  = 11;

int R = 0; int
G = 0;
int B = 0;

void setup() {
  pinMode(RedLed, OUTPUT);
pinMode(GreenLed, OUTPUT);
  pinMode(BlueLed, OUTPUT);
}

void loop() {
  for (int ColorValue = 0; ColorValue <= 255; ColorValue++) {
Conversion(ColorValue);     analogWrite(RedLed, R);
analogWrite(GreenLed, G);     analogWrite(BlueLed, B);
    delay(50);
  }
  for (int ColorValue = 255; ColorValue >= 0; ColorValue--) {
Conversion(ColorValue);     analogWrite(RedLed, R);
analogWrite(GreenLed, G);     analogWrite(BlueLed, B);
    delay(50);
  }
}

int Conversion(int ColorValue) {
if(ColorValue < 85) {     R=255 -
ColorValue * 3;
    G=0;
    B=ColorValue * 3;
  }
  else if(ColorValue < 170) {
ColorValue = ColorValue-85;
    R=0;
    G=ColorValue * 3;
    B=255 - ColorValue * 3;
  }
else {
    ColorValue = ColorValue-170;
    R=ColorValue * 3;
    G=255 - ColorValue * 3;
    B=0;
  }
}
```

Project 41: Een RGB–Led met Rotary Encoder

We zouden die ColorValue ook handmatig in kunnen stellen, b.v. met een potmeter die aan – en + is verbonden. De spanning op de middenaftakking varieert dan van 0 naar Vcc en dat kunnen we inlezen met AnalogRead. Dat levert een waarde van 0 tot 1023. Omzetten naar 0 tot 255 en klaar.

Maar het is wel leuk om het eens anders te doen, n.l. met een Rotary Encoder. Dat is een constructie waarbij 2 outpinnen afwisselend met de middenaansluiting verbonden worden. Het ene contact wordt eerder gemaakt dan het andere. Daaruit is de draairichting af te leiden. Elke keer als er een contact wordt gemaakt wordt de teller met één verhoogd (of verlaagd).
Het omzetten naar een signaal is nu een stuk lastiger.
Een goede uitleg is te vinden op http://bildr.org/2012/08/rotary-encoder-arduino

Essentieel is hier het gebruik van een Interruptroutine.
De Arduino kent 2 interruptpinnen. Elke keer als daar iets gebeurd wordt een interruptroutine aangeroepen. Hier is dat updateEncoder.
Er is een keuze voor een interrupt bij een CHANGE , een LOW, een RISING of een FALLING signaal. Hier gebruiken we de CHANGE.
Omdat 2 en 3 de enige interruptpinnen van een Uno zijn móet de encoder hier op aangesloten worden!
Verbind de middenaansluiting met Gnd.
De RotaryEncoder heeft ook een Switch. Die gebruiken we om de LED's uit te zetten. Ze gaan aan als er aan het duimwiel wordt gedraaid.
Andere bijzonderheden in deze code is het gebruik van pinMode(encoderPin1, INPUT_PULLUP); Dat spaart weerstandjes naar de + uit. De encoderpinnen worden door de interne weerstand "*naar de plus getrokken*". Zo is de toestand van de pinnen altijd gedefinieerd.
Opnieuw gebruiken we de pinnen 9, 10 en 11 en de vlak daarbij zittende Gnd voor R,G en B-Led.



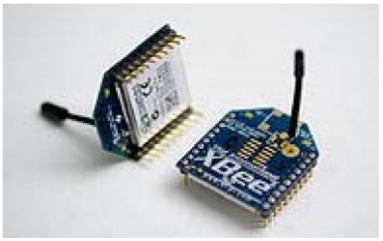
De sketch:

```
int RedLed   = 9; int
GreenLed = 10;
int BlueLed  = 11;

int R = 255; int
G = 0;
int B = 0;

int encoderPin1 = 2; //Wit.   This pin can not be changed!  2/3 are special pins int
encoderPin2 = 3; //Grijs. This pin can not be changed!  2/3 are special pins int
switchPin     = 4;  // Paars

volatile int lastEncoded = 0;
volatile long encoderValue = 0;

void setup() {
  Serial.begin (9600);
  pinMode(encoderPin1, INPUT_PULLUP);//turn pullup resistor on
pinMode(encoderPin2, INPUT_PULLUP);//turn pullup resistor on
pinMode(switchPin,   INPUT_PULLUP);//turn pullup resistor on
pinMode(RedLed, OUTPUT);   pinMode(GreenLed, OUTPUT);
  pinMode(BlueLed, OUTPUT);

  attachInterrupt(0, updateEncoder, CHANGE);
attachInterrupt(1, updateEncoder, CHANGE);  }

void loop() {
  if (digitalRead(switchPin) == LOW)   { // Zet de led's uit
analogWrite(RedLed, 0);     analogWrite(GreenLed, 0);
    analogWrite(BlueLed, 0);
  }
}

void updateEncoder() {
  int MSB = digitalRead(encoderPin1); //MSB = most significant bit   int
LSB = digitalRead(encoderPin2); //LSB = least significant bit

  int encoded = (MSB << 1) | LSB; //converting the 2 pin value to single number   int
sum  = (lastEncoded << 2) | encoded; //adding it to the previous encoded value

  if (sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011) encoderValue ++;
if (sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000) encoderValue --;   if
(encoderValue > 255) encoderValue = 255;   if (encoderValue < 0) encoderValue = 0;
  lastEncoded = encoded; //store this value for next time

  Conversion(encoderValue); // Deze functie verder identiek aan die van RGB_Auto Fader
```

Project 42: Een RGB–Led via Bluetooth met een app op Android Smartphone

Iets op afstand bedienen is '*Magic'*. Hoe bedienen we op afstand een Arduino met daarop aangesloten Led's, motoren enz. enz.? (Voor b.v. een op afstand bestuurbare race auto)
En hoe krijgen we informatie van een Arduino terug? (Om b.v. een weerstation te maken met temperatuurgrafiek op de PC?)

Er zijn verschillende mogelijkheden….
Met Xbee radio modules een twee-weg verbinding opbouwen. Zulke modules zijn prijzig.



Losse radiomodules gebruiken zoals de RFM12 chips en modules. Dat komt neer op een Xbee "*zelf doen*". Met zulke modules haal je een afstand van ca. 30 meter. Ze werken op 868 MHz. (Daar zijn vrij beschikbare banden).

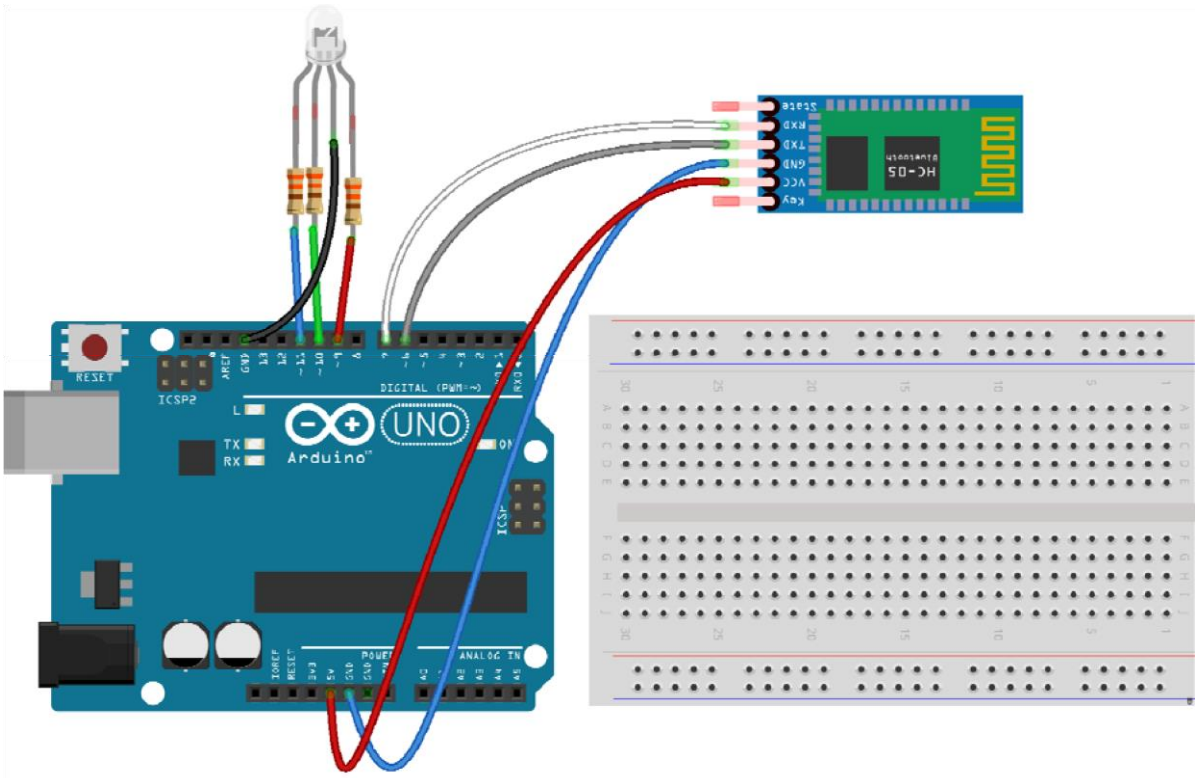
Op korte afstand werkt Bluetooth ook goed. Bluetooth is een radioverbinding in de 2,4GHz-band. Het is geschikt voor communicatie over relatief korte afstanden.
Er zijn verschillende kleine en goedkope BlueTooth ontvangers te koop. De HC-05 is er zo eentje. Een Arduino met zo'n module kan Bluetooth signalen ontvangen. Maar waarmee zenden we die signalen uit? Het leukste is met een Smartphone. Maar dan moet die Smartphone wel daarvoor geprogrammeerd worden.
Dat is tegenwoordig best zelf te doen, b.v. Met AppInventor van MIT.
Zie http://appinventor.mit.edu/explore/
Omdat het hier alleen gaat om een 1e kennismaking gebruiken we een vrij beschikbare app om een RGB Led te bedienen via Bluetooth. Deze app is te vinden op
https://play.google.com/store/apps/details?id=appinventor.ai_yuanryan_chen.BT_LED

En redelijke uitleg is ook te vinden op https://www.instructables.com/id/Android-Controlled-RGB-LEDUsing-Arduino/

In *Bluetooth_RGB_LED_Controller* is opnieuw de RGB Led gedefinieerd op de pinnen 8,9, 10 en 11 (Gnd, Red, Green, Blue)

Een probleem bij het aansluiten van Bluetooth modules is dat ze communceren met de Arduino via de Rx en Tx lijn. (Arduino pin 0 en pin 1) Maar een Uno heeft één Seriele poort en Rx en Tx worden óók gebruikt om een programma te uploaden of data op de Seriele monitor weer te geven.
Dat lukt dus niet met een Bluetooth module, aangesloten op Rx en Tx, pin 0 en 1.

Een oplossing is het gebruik van SoftwareSerial. Daarmee kan een Seriële poort op andere pinnen gecreëerd worden.



De sketch bevat een aantal essentiële regels. (De andere zijn uiteraard ook belangrijk…)

Hier wordt een seriële poort gemaakt op 6 en 7.

```
#include <SoftwareSerial.h>
#include <Wire.h>
SoftwareSerial BT(6, 7); // RX is pin 6, connect to TX of HC-05 BT device = grijs
// TX is pin 7, connect to RX of HC-05 BT device = wits
```

Hier worden enkele Strings gedefineerd. Via de seriële verbinding komt uit de Smartphone App niets meer of minder dan een string van characters binnen.

```
String RGB = ""; //store RGB code from BT
String RGB_Previous = "255.255.255)";
String ON = "ON"; //Check if ON command is received String
OFF = "OFF"; //Check if OFF command is received boolean
RGB_Completed = false;
```

Hier worden de seriele poorten gedefinieerd.

```
 Serial.begin(9600); //Arduino serial port baud rate : 9600
 BT.begin(9600);     //HC-05 module default baud rate is 9600
 RGB.reserve(30);    // Maakt een ruimte van 30 characters in de string RGB
```

Hier wordt de seriële poort gelezen.

```
 while (BT.available()) {     char ReadChar = (char)BT.read();
   if (ReadChar == ')') RGB_Completed = true;  // Right parentheses )
indicates string is complete     else RGB += ReadChar;
```

Hier wordt de string op de monitor geprint als test en check.

```
 if (RGB_Completed) {
   //Print out debug info at Serial output window
   Serial.print("RGB received:");
   Serial.print(RGB);
   Serial.print("     Previous RGB:");
   Serial.println(RGB_Previous);
```

Hier wordt de string RGB geïnterpreteerd. De string kan “ON” , “OFF” zijn of een kleurwaarde.

```
   if (RGB == ON) {
digitalWrite(13, HIGH);
     RGB = RGB_Previous; //We only receive 'ON', so get previous RGB color back to turn LED on
Light_RGB_LED();

   } else if (RGB == OFF) {
digitalWrite(13, LOW);
     RGB = "0.0.0)"; //Send OFF string to turn light off
Light_RGB_LED();
```

Als de string RGB een kleurwaarde bevat wordt die geïnterpreteerd met de functie Light_RGB_LED.

```
    Light_RGB_LED();
    RGB_Previous = RGB;
```

Dat interpreteren gaat als volgt:

```
void Light_RGB_LED() {   int SP1 =
RGB.indexOf('.');   int SP2 =
RGB.indexOf('.', SP1 + 1);   int SP3 =
RGB.indexOf('.', SP2 + 1);
  String R = RGB.substring(0, SP1);
  String G = RGB.substring(SP1 + 1, SP2);
  String B = RGB.substring(SP2 + 1, SP3);
```

En uiteindelijk worden de RGB waarden naar de LED’s gestuurd. Het zijn strings. Ze moeten nog even omgezet worden…..

```
 analogWrite(RedLed,     (R.toInt()));
analogWrite(GreenLed,    (G.toInt()));
analogWrite(BlueLed,  (B.toInt()));
```

Op de Android Smartphone moet dit Bluetooth device eerst “*gepaired*” worden.
Kies Instellingen, Bluetooth, Scannen en selecteer de BT4 module. Het password is altijd “1234”.
Nu is er een verbinding en kan in de app via de opdracht “BT List” dit Bluetooth device gekozen worden.

[Er is een gecompliceerde truc om een Bluetooth device een unieke naam te geven. Eén van de pennen moet hoog gemaakt worden vóór de voeding aangesloten wordt. Dat brengt de module in AT-mode. En dan is een Rename commando mogelijk. Lastig maar dit is hoe deze module aan z’n naam komt…]

Project 43: Een RGB –Led met een Bluetooth Shield op Android Smartphone

Sketch: *Bluetooth_RGB_LED_Controller_Itead_Shield*

Een (klein) bezwaar van losse HC-05 modules is dat er 4 verbindingen gemaakt moeten worden met de Arduino. Al die verbindingen (via Breadboard of direct) zijn kwetsbaar. Een alternatief is het gebruik van een Bluetooth stackable shield.
Zo'n shield is onvermijdelijk direct verbonden met Rx en Tx op pin 0 en 1. Het bezwaar ervan is dat uploaden niet meer lukt want de seriële verbinding is daarvoor niet meer beschikbaar.

Maar…. dit shield is makkelijk te hacken. In de gemodificeerde vorm wordt de Rx/Tx verbinding met het shield verbroken in de hierboven getekende stand van de buitenste schakelaar. Dit is dus de stand voor het uploaden van code. In de andere stand werkt het shield zoals bedoeld.

Klein bezwaar: de USB-Serial is intern op een Uno verbonden met pin 0 en 1. We zijn dus de mogelijkheid kwijt om via de seriële monitor informatie terug te krijgen.

Het uploaden van de sketch gaat als vanouds.
Maar op de Smartphone moet dit Bluetooth device eerst "*gepaired*" worden.
Instellingen, Bluetooth, Scannen en selecteer de "*itead*" module. Het password is altijd "1234".
Nu is er een verbinding en kan in de app via de opdracht "BT List" dit Bluetooth device gekozen worden.

```
int RedLed = 9;   int GreenLed = 10;   int BlueLed = 11;   int RED_LED = 13;

String RGB = ""; //store RGB code from BT
String RGB_Previous = "255.255.255)"; String ON = "ON"; //Check if ON command is received
String OFF = "OFF"; //Check if OFF command is received boolean
RGB_Completed = false;

void setup() {
  Serial.begin(9600);     //HC-05 module default baud rate is 9600   RGB.reserve(30);
  pinMode(RED_LED, OUTPUT);//Set pin13 as output for LED,
}

void loop() {
  while (Serial.available()) {    //Read each character from Serial Port(Bluetooth)
char ReadChar = (char)Serial.read();
    if (ReadChar == ')') RGB_Completed = true;  // Right parentheses ) indicates string is complete
else RGB += ReadChar;
  }

  if (RGB_Completed) {    //When a command code is received completely with ')' ending character
if (RGB == ON) {       digitalWrite(13, HIGH);
     RGB = RGB_Previous; //We only receive 'ON', so get previous RGB color back to turn LED on
Light_RGB_LED();

    } else if (RGB == OFF) {
digitalWrite(13, LOW);
     RGB = "0.0.0)"; //Send OFF string to turn light off
     Light_RGB_LED();
    } else {
     Light_RGB_LED();       //Turn the color according the color code from Bluetooth Serial Port
RGB_Previous = RGB;
    }
    RGB = "";     //Reset RGB String
    RGB_Completed = false;
  }
}

void Light_RGB_LED() {   int SP1 =
RGB.indexOf('.');   int SP2 =
RGB.indexOf('.', SP1 + 1);   int SP3 =
RGB.indexOf('.', SP2 + 1);
  String R = RGB.substring(0, SP1);
  String G = RGB.substring(SP1 + 1, SP2);
  String B = RGB.substring(SP2 + 1, SP3);

  analogWrite(RedLed,      (R.toInt()));
analogWrite(GreenLed,    (G.toInt()));
analogWrite(BlueLed,  (B.toInt())); }
```

Project 44: Inputs: een Stackable Joystick Shield met buttons

Sketch: *Joystick_RGB_Led_no_delays*

Joysticks bestaan uit twee haaks op elkaar staande potmeters, geschakeld als spanningsdeler tussen de Gnd en de Vcc.
De beweging van de joystick leidt tot een andere uitgangsspanning.
Een handige manier om een joystick aan te sluiten is het stackable joystick shield van SparkFun.
Behalve de joystick met twee potmeters en een drukschakelaar zijn er ook 4 extra buttons beschikbaar.
De potmeters zijn verbonden met Analog 0 en Analog 1, de schakelaars zijn verbonden met D3 t/m D7.
(Oorspronkelijk D2 t/m D6 maar deze kleine wijziging bleek handig in sommige toepassingen.)

Joysticks zijn geen erg nauwkeurige input devices.
Het is een pot meter verbonden met – en +. Het shield verbind de X en Y potmeter met A0 en A1. We lezen dus in met een analogRead(PIN_ANALOG_X); Dat levert een waarde op tussen de 0 en de 1023.
Om dat om te zetten naar een wat handiger signaal wordt het commando map(Xmeet, 0, midX, -100, 0); gebruikt. Dat 'mapped' de invoerwaarde Xmeet die tussen de 0 en de midX varieert lineair naar een output value tussen de -100 en de 0.
Op deze manier wordt bereikt dat we en horizontaal en verticaal een joystick signaal krijgen dat van -100 naar 100 loopt met daarbij een kleine instelbare Threshold.

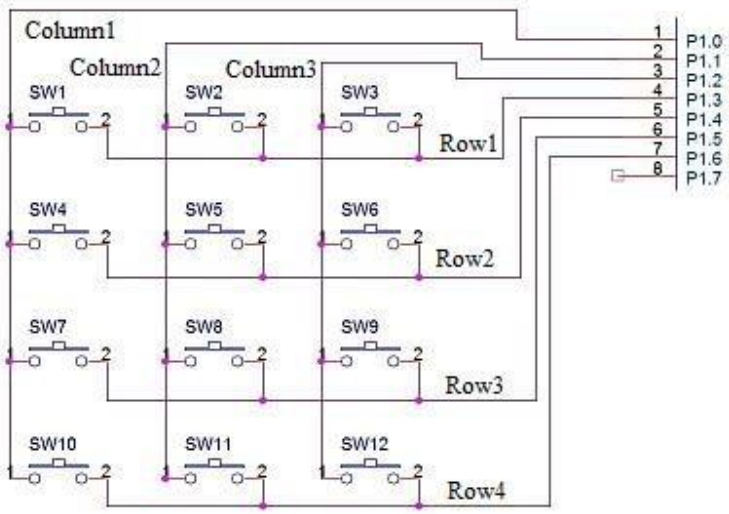
Natuurlijk is het uitlezen van joystick en buttons weer geregeld met een "timer" constructie.
Elke 100 msec. Wordt het shield uitgelezen. Dat is 10x per seconde. Waarom zou je het vaker doen?

De schakelaars worden ook ingelezen en sturen in dit voorbeeld de RGB_Led aan.
Linker button = Rood_Aan, Rechter button = Rood_Uit
Up button = Groen_Aan, Down button=Groen_Uit
Joystickbutton pressed= Blue_Aan
Het joysticksignaal zien we pas als we de Seriële Monitor openen..

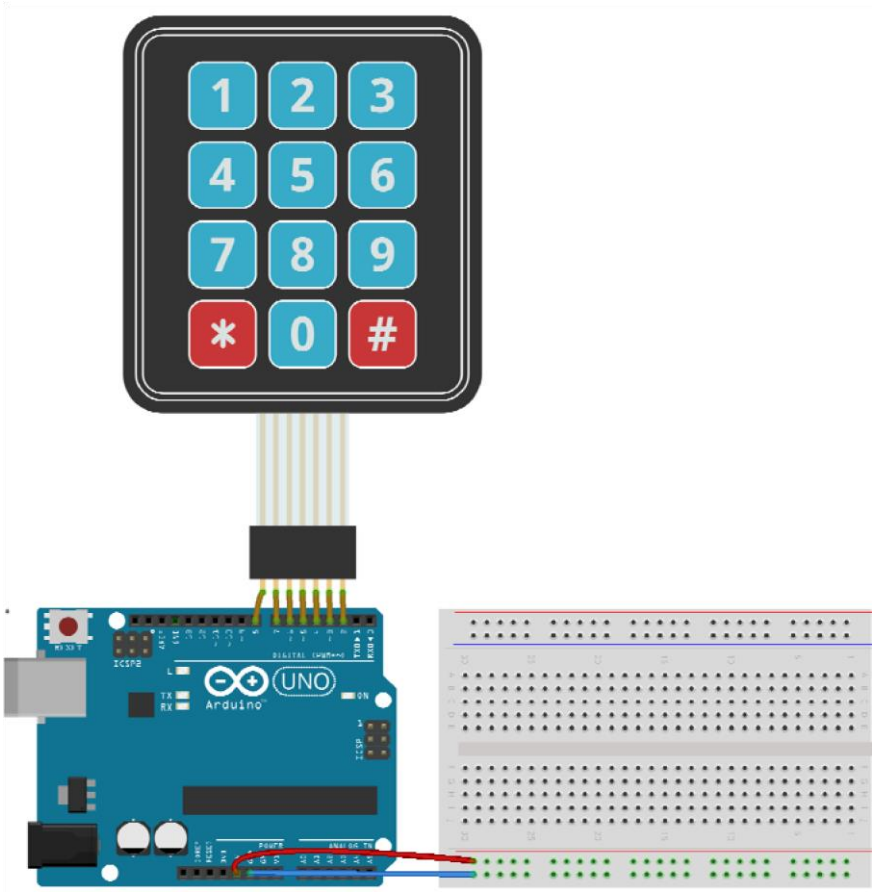
```
const byte PIN_BUTTON_RIGHT   = 3;   const byte PIN_BUTTON_UP         = 4; const
byte PIN_BUTTON_DOWN   = 5;   const byte PIN_BUTTON_LEFT      = 6; const byte
PIN_BUTTON_SELECT = 7;
const byte PIN_ANALOG_X = 0;   const byte PIN_ANALOG_Y = 1; int
RedLed   = 9;   int GreenLed = 10;   int BlueLed  = 11;

int Threshold = 2; // De joystick is erg gevoelig.... int
midX = 0;   int midY = 0;  int Xval = 0;   int Yval = 0;

const int Reading_Period = 100;          // Update reading every 100 msec.
unsigned long time_to_Read = 0;

void setup() {
Serial.begin(9600);
  pinMode(RedLed, OUTPUT);     pinMode(GreenLed, OUTPUT);     pinMode(BlueLed, OUTPUT);
pinMode(PIN_BUTTON_RIGHT,  INPUT_PULLUP);     pinMode(PIN_BUTTON_LEFT,
INPUT_PULLUP);   pinMode(PIN_BUTTON_UP,         INPUT_PULLUP);
pinMode(PIN_BUTTON_DOWN,  INPUT_PULLUP);
pinMode(PIN_BUTTON_SELECT,INPUT_PULLUP);

  midX = analogRead(PIN_ANALOG_X); // middenpositie van de Joystick
midY = analogRead(PIN_ANALOG_Y); // middenpositie van de Joystick }

void loop() {
Joystick_Reader();
}

void Joystick_Reader() {
  if ((millis() - time_to_Read) > Reading_Period)  {
    if (digitalRead(PIN_BUTTON_LEFT)  == LOW)  digitalWrite(RedLed, HIGH);     if
(digitalRead(PIN_BUTTON_RIGHT) == LOW)  digitalWrite(RedLed, LOW);     if
(digitalRead(PIN_BUTTON_UP)    == LOW)  digitalWrite(GreenLed, HIGH);     if
(digitalRead(PIN_BUTTON_DOWN)  == LOW)  digitalWrite(GreenLed, LOW);     if
(digitalRead(PIN_BUTTON_SELECT)== LOW)  digitalWrite(BlueLed, HIGH);     if
(digitalRead(PIN_BUTTON_SELECT)== HIGH) digitalWrite(BlueLed, LOW);

    int Xmeet = analogRead(PIN_ANALOG_X);     if (Xmeet >= midX +
Threshold) Xval   = map(Xmeet, midX, 1023, 0, 100);     if (Xmeet <  midX -
Threshold) Xval   = map(Xmeet, 0, midX, -100, 0);     if (Xmeet <  midX +
Threshold && Xmeet >  midX - Threshold) Xval   = 0;

    int Ymeet = analogRead(PIN_ANALOG_Y);     if (Ymeet >= midY +
Threshold) Yval   = map(Ymeet, midY, 1023, 0, 100);     if (Ymeet <  midY -
Threshold) Yval   = map(Ymeet, 0, midY, -100, 0);     if (Ymeet <  midY +
Threshold && Ymeet >  midY - Threshold) Yval   = 0;

    Serial.print("x:");       Serial.print(Xval);       Serial.print(" ");
    Serial.print("y:");       Serial.print(Yval);       Serial.print(" ");       Serial.println();
time_to_Read = millis();
  }
}
```

Project 45: Inputs: een membraan keypad

Een ander “Input-device” is een Matrix membraan Keypad. Dat zijn eenvoudige folieschakelaars, gecombineerd tot rijen en kolommen.



Zolang je niet meerdere schakelaars tegelijk indrukt komt er op de connector een éénduidig signaal uit. Maar omzetten naar een werkelijke toets is nog wel lastig. Gelukkig is daar de Keypad.h library voor. Met een klein beetje voorizchtigheid is het keypad rechtstreeks op de Arduino “in te prikken” op de pennen 8-7-6-5-4-3-2. (Pen 8 is de uitgebogen pen ….)



De weergave is zoals altijd weer op het I2C LCD display.

```
//Inputs: een Matrix Keypad

#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address

#include <Keypad.h> const byte
ROWS = 4; //four rows const byte
COLS = 3; //three columns
char keys[ROWS][COLS] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};
byte rowPins[ROWS] = {8, 7, 6, 5}; //connect to the row pinouts of the keypad byte
colPins[COLS] = {4, 3, 2}; //connect to the column pinouts of the keypad

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

void setup() {
lcd.begin(16, 2);
lcd.setCursor(0, 0);
  lcd.print("Matrix Keypad");
}

void loop() {
  char key = keypad.getKey();
  if (key) {
lcd.setCursor(0,1);
lcd.print(key);
    lcd.print(" pressed!");
  }
}
```

Project 46: Toegangscontrole met membraan keypad

Zo'n matrix keypad vráágt natuurlijk om een toepassing in een project waar iets met een pincode ontgrendeld moet worden. Dit programma zet even een led op pin 9 aan als de juiste code is ingevoerd. Maar dat gaat niet helemaal vanzelf….
Zo´n code begint een keer en moet ook binnen een bepaalde tijd worden afgemaakt. Vanaf de 1e keypress start een timer die bijhoudt of ook de 2e, 3e en 4e keypress wel binnen de tijd zijn gedaan.
Zo niet, dan volgt de melding lcd.print("Te langzaam.....").
Indien wel binnen de tijd moet daarna die code gecheckt worden tegen de echte code.
Klopt dat niet, dan volgt de melding lcd.print("Niet correct.
Klopt het wel, dan volgt eindelijk de melding lcd.print("Correct!... ");!"
(En gaat de led even aan. Maar dat zou vervangen kunnen worden door een elektromagnetisch slot) En daarna begint alles weer van voor af aan. Weergave zoals altijd weer op een I2C display.

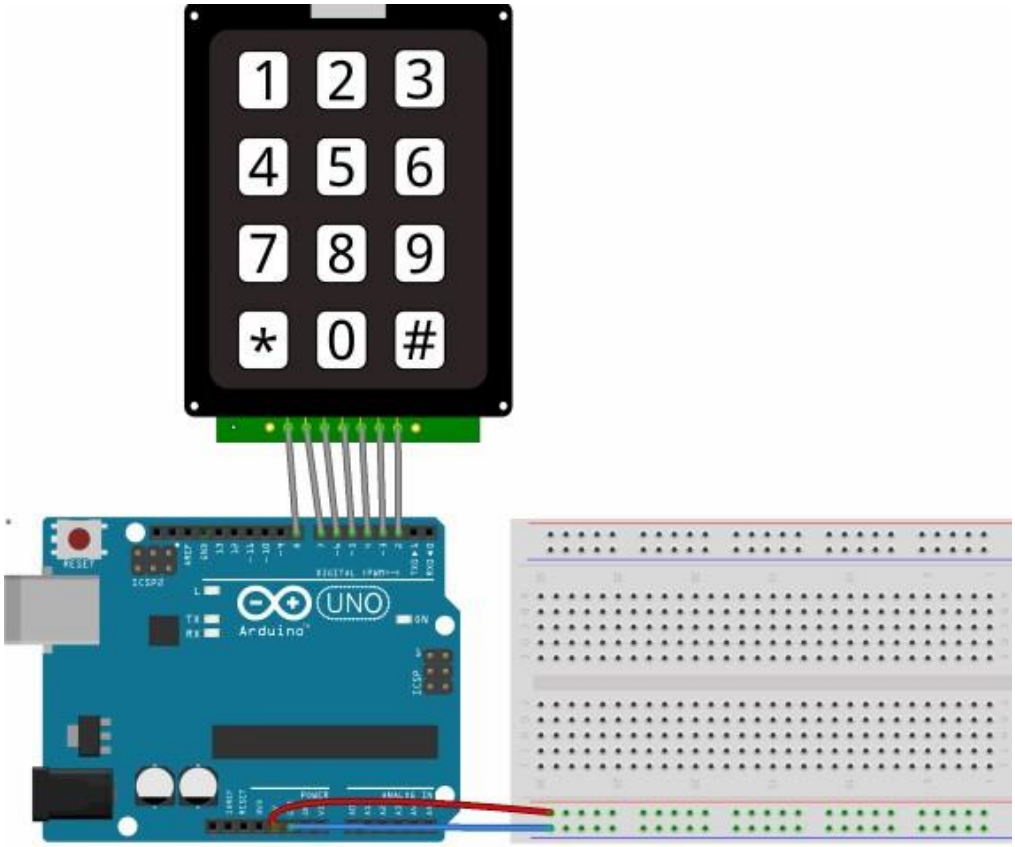
```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address

#include <Keypad.h> const byte
Rows = 4; //four rows const byte
Cols = 3; //three columns
char Keys[Rows][Cols] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};

byte rowPins[Rows] = {8, 7, 6, 5}; //connect to the row pinouts of the keypad byte
colPins[Cols] = {4, 3, 2}; //connect to the column pinouts of the keypad
Keypad mijn_keypad = Keypad( makeKeymap(Keys), rowPins, colPins, Rows, Cols); char
key;

char unlockcode[] = {'1', '2', '3', '4'};  // De defaultcode=1234 char
invoercode[] = {'-', '-', '-', '-'};  // De ingevoerde code

int pincijfers = 0; boolean
pincorrect = LOW; unsigned
long time0; unsigned long
timeout = 4000;
int ledpin = 9;

void setup() {
Serial.begin(9600);
pinMode(ledpin, OUTPUT);
  lcd.begin(16, 2);                    // Initialiseer het display   lcd.setCursor(0,
0);
  lcd.print("Enter Pincode..."); }
```

```
void loop() {   key = mijn_keypad.getKey();      // Lees keypress in en handel de numerieke
keyinvoer af.   if (key && pincijfers == 0 ) {   // Er is een nummerkey ingedrukt en de code is nog
niet volledig     time0 = millis();     invoercode[0] = key;     lcd.setCursor(pincijfers, 1);
   lcd.print("*"); // of juist wel de key weergeven met lcd.print(invoercode[0]);
pincijfers = 1;
  }
  else {  // Er is binnen de timeout een key ingedrukt en de code is nog niet volledig
if (key && pincijfers > 0 && pincijfers <= 3 && (millis() - time0) < timeout) {
invoercode[pincijfers] = key;       lcd.setCursor(pincijfers, 1);
     lcd.print("*"); // of juist wel de key weergeven met lcd.print(invoercode[pincijfers]);
pincijfers = pincijfers + 1;
     if (pincijfers == 4) { // Houd die vierde input door een delay kort zichtbaar
delay(400);         lcd.setCursor(0, 1);         lcd.print("               ");
     }
    }
  }
  if (millis() - time0 > timeout && pincijfers != 0) { // De timeout is verstreken zonder volledige code
for (int i = 0; i <= 3; i++) invoercode[i] = '-';     pincijfers = 0;     lcd.setCursor(0, 1);     lcd.print("
"); // Haal direct de code weg.     lcd.setCursor(0, 0);
   lcd.print("Te langzaam....."); // Geef een message
digitalWrite(ledpin, LOW);     delay(2000);
lcd.setCursor(0, 0);
   lcd.print("Enter Pincode..."); // Herstel de tekst op het scherm
  }
  if (pincijfers == 4) { // Er is een 4-cijferige code ingevoerd
   if (invoercode[0] == unlockcode[0] && invoercode[1] == unlockcode[1] && invoercode[2] == unlockcode[2]
                                        && invoercode[3] == unlockcode[3] ) {pincorrect = HIGH;}
if (pincorrect == LOW) { // Dus de ingevoerde code is niet de juiste code..
     pincijfers = 0;                    //reset ingevoerde code
lcd.setCursor(0, 0);
     lcd.print("Niet correct!..."); // Herstel de tekst op het scherm
delay(2000);       lcd.setCursor(0, 0);
     lcd.print("Enter Pincode..."); // Herstel de tekst op het scherm
   }
   if (pincorrect == HIGH) { //  De juiste pincode!!!  Doe nu iets!
     lcd.setCursor(0, 0);
     lcd.print("Correct!...     "); // Herstel de tekst op het scherm
digitalWrite(ledpin, HIGH); // Hier kan ook een andere actie volgen
delay(2000);       pincijfers = 0; //reset ingevoerde code
     pincorrect = LOW;
digitalWrite(ledpin, LOW);
     lcd.setCursor(0, 0);
     lcd.print("Enter Pincode..."); // Herstel de tekst op het scherm
   }
  }
}
```

Project 47: Inputs: Druktoets keypad

Iets mooier dan een membraan keypad is een echt keypad met druktoetsen.
Het membraan keypad heeft een andere volgorde van de pinnen dan het druktoetsen keypad.
Het is nogal een gepuzzel om de aansluitingen zó te krijgen dat het handig aansluit.
Dat is hier gelukt. De verbindingen lopen nu netjes naast elkaar. Om zó te kunnen aansluiten én de ingedrukte toets te kunnen herkennen zijn de volgende twee regels in het programma essentieel.

```
byte rowPins[Rows] = {7, 2, 3, 5}; // connect to the row pinouts of the keypad Row1,Row0, byte
colPins[Cols] = {6, 8, 4};        // connect to the column pinouts of the keypad
```

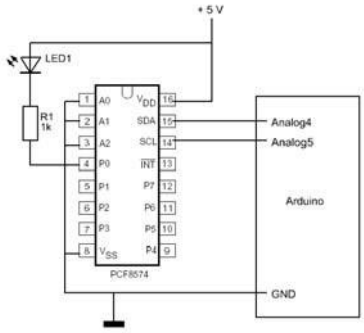


Het programma maakt weer gebruik van weergave op het I2C LCD display.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address

#include <Keypad.h>

const byte Rows = 4; // four rows
const byte Cols = 3; // three columns

char keys[Rows][Cols] = {
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};

// Keypad 1 Not Connected, Keypad 2 aan Arduino8, Keypad 3 aan Arduino7, Keypad 4 aan Arduino6,
// Keypad 5 aan Arduino5, Keypad 6 aan Arduino4, Keypad 7 aan Arduino3, Keypad 8 aan Arduino2

byte rowPins[Rows] = {7, 2, 3, 5}; // connect to the row pinouts of the keypad Row1,Row0, byte
colPins[Cols] = {6, 8, 4};    // connect to the column pinouts of the keypad

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, Rows, Cols );

void setup() {
Serial.begin(9600);   lcd.begin(16,
2);   lcd.setCursor(0, 0);
  lcd.print("Druktoets Keypad");
}

void loop() {
  char key = keypad.getKey();

  if (key) {
lcd.setCursor(0, 1);
lcd.print(key);
    lcd.print(" pressed!");
  }
}
```

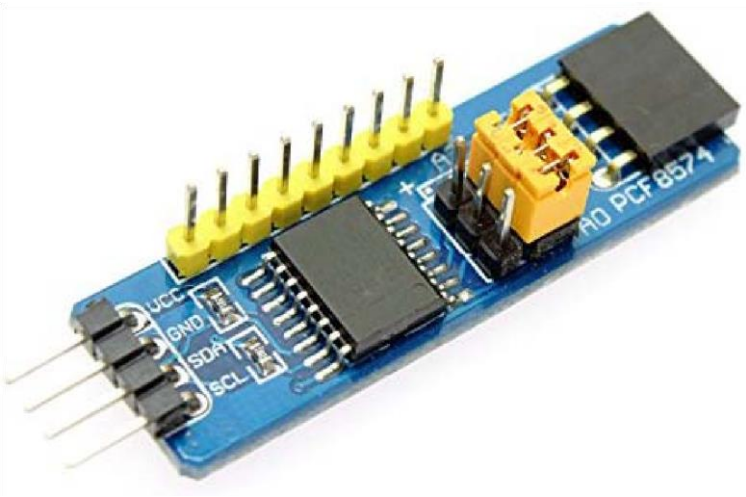
Project 48: Meer inputs en outputs met een PCF8574 I2C port expander

In de vorige sketch werd een druktoetsen keypad uitgelezen met een PCF8574 Port Expander. Geniaal ICtje. Ineens wordt het op relatief hoge snelheden mogelijk om véél meer in en uitgangen te creeeren via een eenvoudige 2 draads aansluiting via de I2C bus op A4 en A5.



Dat idee is mooi uitgewerkt in de PCF8574 I2C to 8-Bit Digital Port Expander van Hobbyelectronics.
Elke Port Expander kan met jumpers op een uniek adres ingesteld worden.
En deze Port Expanders kunnen direct op elkaar doorgekoppeld worden. (Mits op een uniek adres ingesteld).
Op deze manier kunnen maximaal 8 expanders gekoppeld worden zodat er 64 in of uitgangen beschikbaar zijn.

A2	A1	A0	adres
L	L	L	0x20
L	L	H	0x21
L	H	L	0x22
L	H	H	0x23
H	L	L	0x24
H	L	H	0x25
H	H	L	0x26
H	H	H	0x27



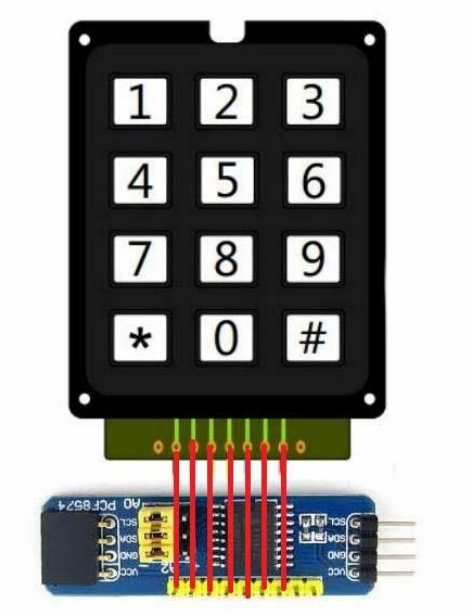
Hier worden één schakelaar en 2 led’s aangesloten op resp. 5, 6 en 7 (en Gnd)

Er gebeurd in dit programma niet veel bijzonders. Als de schakelaar wordt ingedrukt gaan de led's knipperen.
Het inlezen en uitlezen van deze uitbreidingspinnen is niet erg ingewikkeld. Zoals vaker is het voor de leesbaarheid van een programma best handig om gebruik te maken van libraries waar alle functionaliteit al in ondergebracht hebben. Zoals in de HCPCF8574 library. (Waar ook een voorbeeld sketch te vinden voor twee gekoppelde expanders).
Sluit de SDA lijn aan op A4 (Oranje=SDA) en de CLK lijn op A5 (Geel=CLK)
Sluit een schakelaar en 2 led's aan op pin 5, 6 en 7. Stel de jumpers in op het gewenste adres en gebruik dat in de sketch.

```
// Meer inputs en outputs met een PCF8574 I2C port expander

#include "HCPCF8574.h"          //Include the HCPCF8574 library byte
I2C_ADD = 0x20;             //I2C address of the PCF8574 HCPCF8574
Port(I2C_ADD);   //Create an instance of the library


int PortLedpin1 = 6; int
PortLedpin2 = 5;
int Portbutton1 = 7;

void setup() {
  Port.init();                                  //Initialize the PCF8574 (all pins are set to high)
  Port.pinMode(Portbutton1, INPUT);      //Set digital pin Portbutton1 to an input
  Port.pinMode(PortLedpin1, OUTPUT); //Set digital pin PortLedpin1 to an output
  Port.pinMode(PortLedpin2, OUTPUT); //Set digital pin PortLedpin1 to an output
  Port.pinWrite(PortLedpin1, HIGH);       //De Led is geschakeld tussen uitgang en de Vcc . Aan als uitgang laag.
Port.pinWrite(PortLedpin2, HIGH);
}

void loop()
{
  if (Port.pinRead(Portbutton1) == LOW) {
  Port.pinWrite(PortLedpin1, LOW);
Port.pinWrite(PortLedpin2, LOW);     delay(500);
  Port.pinWrite(PortLedpin1, HIGH);
Port.pinWrite(PortLedpin2, HIGH);     delay(250);
  }

}
```

Project 49: Inputs: Druktoetskeypad met een PCF8574 I2C port expander

Er zijn 7 pinnen nodig om een keypad aan te sluiten. Dat kan een probleem zijn als er veel in-out pinnen nodig zijn. Opnieuw brengt I2C de oplossing. De I2C port expander is goed te combineren met het druktoets keypad. Wanneer dat aan de onderzijde van een female header wordt voorzien kan het keypad in één keer op de port expander gedrukt worden. Van het keypad is pin 1 NC (Not Connected)
Pin 2 t/m pin 8 komt in P0 t/m P6 van de expander. P7 en Int blijven vrij.
Het resultaat: een I2C druktoetsen keypad.



Bij het testen van dit programma met uitsluitend output naar de seriele monitor gaat het na enkele toetsaanslagen fout!
De oorzaak is dat de SDA lijn zonder extra voorzorgen ‘floating’ is. Er is een pull up weerstand van 4K7 nodig. Zonder pull up weerstand werkt de I2C bus niet goed meer.
Sommige I2C shields –zoals het Backpack LCD shield hebben al zulke pull up weerstanden. Daarom gaat het wel goed als –zoals in onderstaand programma- de port expander samen met het Backpack LCD shield op de I2C bus is aangesloten.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Set the LCD I2C address #include
<i2ckeypad.h>
// Pin 2-3-4-5-6-7-8 connects to P0-P1-P2-P3-P4-P5-P6. P7 and Int Not Connected

#define ROWS 4
#define COLS 3
#define PCF8574_ADDR 0x20 // With A0, A1 and A2 of PCF8574 to ground I2C address is 0x20 i2ckeypad
kpd = i2ckeypad(PCF8574_ADDR, ROWS, COLS);

void setup()
{
  lcd.begin(16, 2);
lcd.setCursor(0, 0);
  lcd.print("I2C Druktoetspad");
  Wire.begin();
kpd.init();
}

void loop()
{
  char key = kpd.get_key();
  if (key != '\0') {
lcd.setCursor(0, 1);
lcd.print(key);
    lcd.print(" pressed!");
  }
}
```

Project 50: Het schrijven en ophalen van gegevens uit EEPROM

In een Arduino Uno is een beetje geheugen beschikbaar. Daar wordt in elk geval het gecompileerde programma opgeslagen. Maar er zijn ook 512 "geheugenplaatsen" waar getallen van 0-255 kunnen worden opgeslagen. Iets opslaan in EEPROM is erg handig want de inhoud blijft bewaard, ook als de Arduino wordt uitgezet! Een handige library waarmee karakters, integers, long integers en zelfs floating getallen door elkaar heen opgeslagen kunnen worden is EEPROMAnything.h
Deze sketch maakt gebruik van een structure. Een structure kan allerlei variabelen bevatten. De volgende sketch schrijft een Long Integer, een Floating Point getal, een gewone Integer en een Boolean in de EEPROM en leest later die gegevens ook uit.

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

struct {   long Long;          //Long variables van -2,147,483,648 to 2,147,483,647.   float
Float;          //Floating-point numbers van -3.4028235E+38 tot 3.4028235E+38.
  int Integer;          //Integers van -32,768 tot 32,767
  boolean Boolean;  //Boolean variabelen zijn false of true, 0 of 1
}
data; // De naam van de 'structure' is ‘data

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

void loop() {
  Serial.println("Defining some values.");
  data.Long = 1234567890;
data.Float = 12345.67890;
data.Integer = 12345;   data.Boolean
= true;
  EEPROM_writeAnything(0, data);
  Serial.println("Writing all values as a structure to EEPROM: ");   delay(10);
  data.Long = 0;       // Delete all values;
data.Float = 0;   data.Integer = 0;
  data.Boolean = false;

  Serial.println("Erasing all values!");
  Serial.println("Reading all values from EEPROM: ");
  EEPROM_readAnything(0, data);
  Serial.print("Long integer= ");
  Serial.println(data.Long);
  Serial.print("Floating = ");
  Serial.println(data.Float, 5);
  Serial.print("Integer = ");
  Serial.println(data.Integer);
  Serial.print("Boolean = ");   Serial.println(data.Boolean);   while (1); // Dit is een
oneindige loop. Hier blijft het programma dus "hangen".
}
```

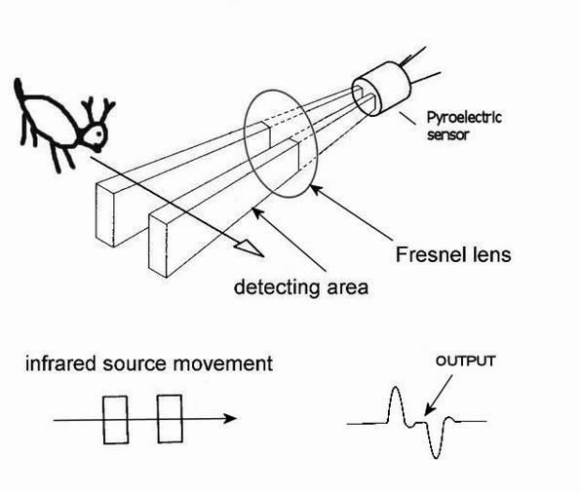
Project 51: Toegangscontrole met Pincode en Admin code

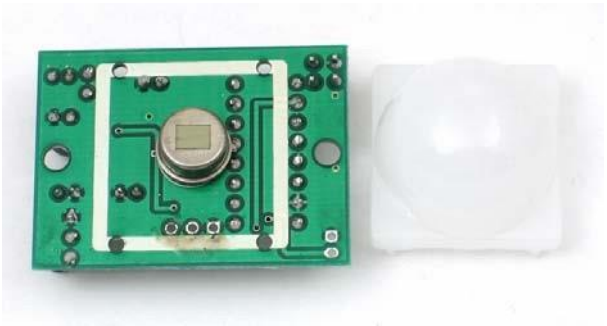
Een kleine toegift.
Dit programma geeft ook de mogelijkheid om een nieuwe pincode in te voeren én te onthouden in EEPROM.
De admin code is in dit programma 13*#
Daarna vraagt het programma om een nieuwe pincode én slaat deze op in EEPROM.
Als output is nog steeds een simpele led gekozen. Maar…. dat kan ook een servomotortje zijn die een slotje bediend. Het resultaat daarvan kan iets zijn zoals op de volgende foto.
Daar komt overigens nog wel wat meer bij kijken. Het deurtje mag niet eerder op slot dan dat het helemaal dicht is. Dus er is ook nog een reedschakelaar in het deurtje en een magneetje in de zijkant opgenomen. En ooit raakt het interne batterijtje leeg. Oplossing: Via de twee koperen spijkertjes links onder kan altijd voeding van buitenaf worden “toegediend”.

Van eerste prototype tot werkend systeem is altijd een lange weg….

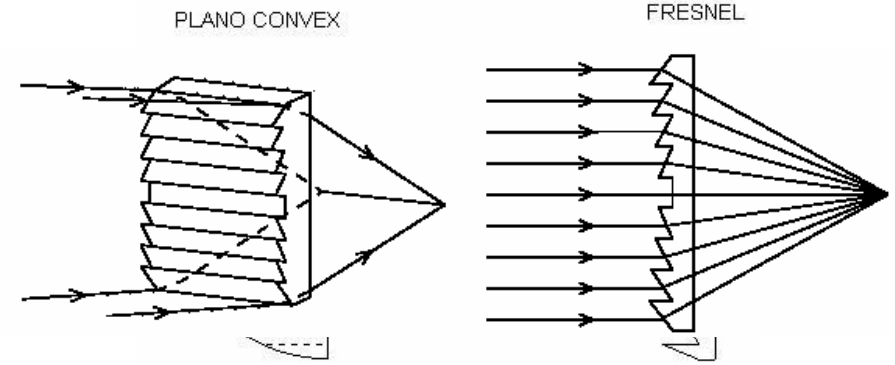
Project 52: Infrarood beweging meting met een PassieveInfraRood sensor

Sketch: *PIR_test1* en *PIR_test2*
Zichtbaar licht kunnen we zien. Maar het is maar een stukje van alle “licht”.
Sommige dieren zien kleuren die wij niet kunnen zien, boven het ‘paars’ , het ‘ultra-violet’.
Of ze zien “infra rood”, dat is wat uitgestraald wordt door elk voorwerp, afhankelijk van de temperatuur. Wij zien dat pas als een voorwerp heel heet is. Dan zien we het rood opgloeien.
Maar daar ‘*onder’* is er Infrarood.
Er bestaan simpele sensoren die Infrarood kunnen meten. En die worden veel gebruikt in alarm installaties. Want een bewegend voorwerp als een inbreker geeft voortdurend infrarode straling af en als we meten of er in een ruimte iets verandert aan het infrarood, dan is er een warm, levend, voorwerp in beweging. Alarm! Zulke sensoren heten Passieve Infrarood sensoren, PIR’s
Een PIR bestaat uit twee sensoren die allebei de hoeveelheid Infrarode straling meten.
Hoe “heter” iemand is, des te meer straling!
Als een warm voorwerp langs een sensor beweegt meet de ene sensor even iets anders dan de andere. Tijd voor een signaal!
De 2 sensoren zijn meestal netjes in één huisje gestopt met een beschermende laagje er over heen.



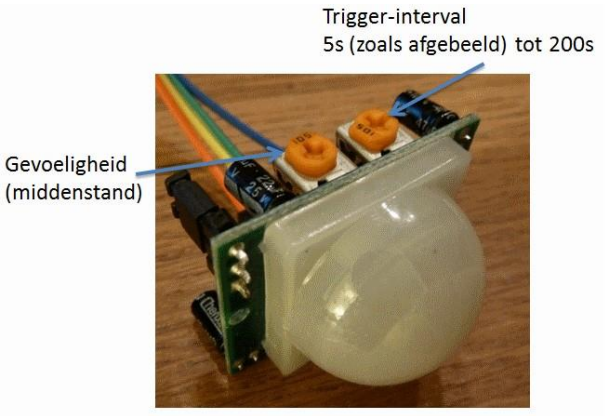


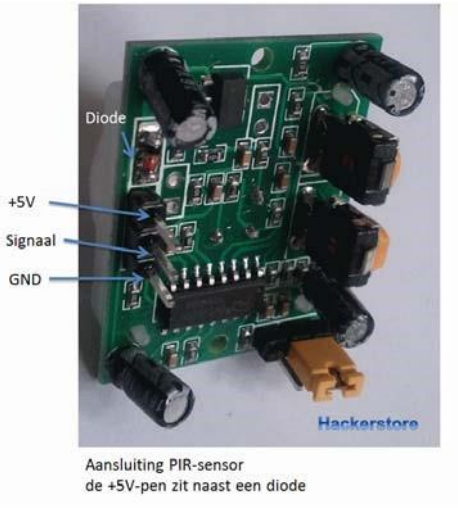
Nu willen we in een brede “bundel” beweging kunnen zien.
Daarvoor moeten de lichtstralen (infrarood is echt gewoon licht!) uit een brede bundel afgebogen worden naar die 2 sensors. Dat vraagt om een hele dikke lens!
In plaats van dikke, dure lenzen wordt bij PIR’s een “*Fresnel*” lens gebruikt. Dat is eigenlijk een dikke lens die in plakjes is gesneden.



De Fresnel lens zorgt dat het licht meer geconcentreert wordt op de sensor. Maar we willen vooral beweging meten. En daarvoor verdelen we nu die Fresnel lens in een aantal segmenten.

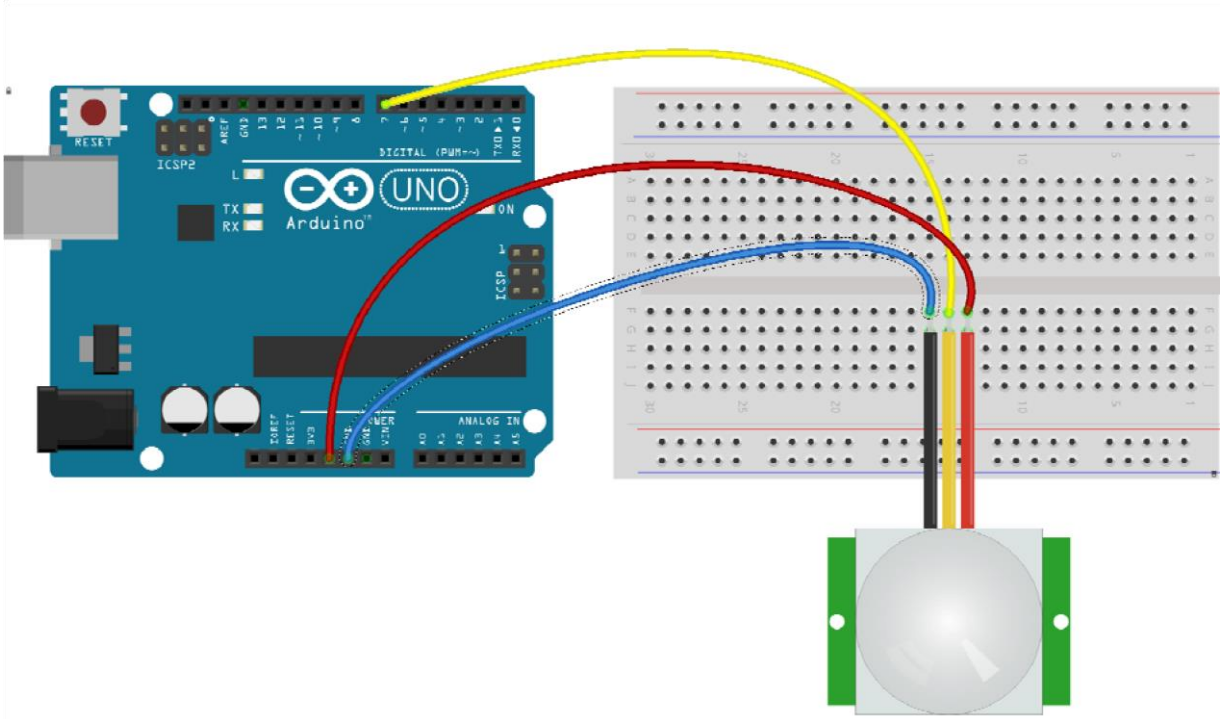
Elk van die stukjes is weer een Fresnel lens.
Er zijn verschillende typen. Ze hebben allemaal een -, een + en een data aansluiting.
Een PIR heeft altijd extra elektronica waarmee een paar dingen geregeld worden.
Als er beweging is moet de PIR reageren. Maar moet die nu voortdurend een signaal afgeven dat er iets beweegt? De meeste PIR's zijn zo gemaakt dat ze niet voortdurend op beweging reageren maar alleen een eerste keer en daarna weer een korte tijd niet….Dat heet het Trigger interval en dat is meestal instelbaar.
En ook de gevoeligheid is in te stellen. De PIR's is al op maximaal gevoelig ingesteld met een klein Trigger interval. (Dat lijkt het beste te werken…)





Aansluiting PIR-sensor
de +5V-pen zit naast een diode

Het aansluiten van een PIR is simpel. Hier wordt de led op pin13 aangezet als er beweging wordt gedetecteerd. Omdat er al allerlei "intelligentie" in de PIR sensor zit is het resultaat soms onverwachts. Het is geen gewone sensor. Pas ná het Triggerinterval reageert de sensor weer…



De sketch:

```
// Infrarood beweging meting met een PIR (PassieveInfraRood)sensor

int ledPin = 13;                // choose the pin for the LED int pirPin =
7;                  // choose the input pin (for PIR sensor) int pirState =
LOW;          // we start, assuming no motion detected int
calibrationTime = 30;

void setup() {
  pinMode(ledPin, OUTPUT);
pinMode(pirPin, INPUT);
  Serial.begin(9600);
  // Give the sensor some time to calibrate
Serial.print("Calibrating sensor ");   for
(int i = 0; i < calibrationTime; i++) {
    Serial.print(".");
delay(100);
  }
  Serial.println(" Done! ");
  Serial.println("Sensor is now active...");   delay(50);
}

void loop() {
  if (digitalRead(pirPin) == HIGH) {  // HIGH dan is er beweging. Turn LED ON
digitalWrite(ledPin, HIGH);
    if (pirState == LOW) {                  // Als dat de eerste keer was, zet dan een message op de monitor
Serial.println("Motion detected!");
      pirState = HIGH;                      // En maak pirState= HIGH
}
  }

  if (digitalRead(pirPin) == LOW) {   // LOW dan is er geen beweging. Turn LED OFF     digitalWrite(ledPin,
LOW);           // turn LED OFF
    if (pirState == HIGH) {                // Als dat na beweging de eerste keer was, zet dan een message op de monitor
      Serial.println("Motion ended!");
pirState = LOW;
    }
}
}
```