2.0. Arduino Programmeren voor Beginners
Deel 2: Uitvoer van informatie

Dit is het tweede deel in een serie lessen voor iedereen die graag zou willen leren programmeren.

In dit deel gaan we de Arduino IDE, die we in deel 1 hebben geïnstalleerd, een beetje verkennen en gaan we kijken hoe we uitvoer (output) van de Arduino zichtbaar kunnen maken op onze computer. Dat laatste is natuurlijk handig als we willen weten of ons programma naar wens werkt, of om tussen door te kijken wat het programma doet. In tegenstelling tot een gewone computer, heeft de Arduino namelijk geen beeldscherm …

Deze les richt zich hoofdzakelijk op het Arduino Programmeren voor beginners – gebrek aan kennis voor wat betreft de Engelse taal en wiskundige achtergrond hoeft waarschijnlijk geen probleem te zijn. Het gebruik van extra elektronica componenten blijft beperkt tot een minimum en bewaren we voor een volgende reeks.

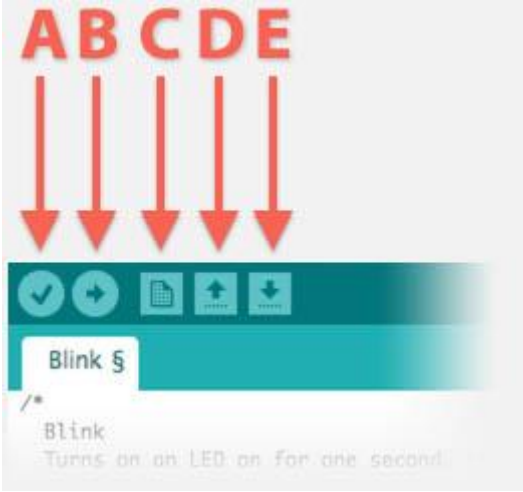
Inhoudsopgave

2.1. Een korte verkenning van de Arduino IDE

Ik ga ervan uit dat je Deel 1 netjes gevolgd hebt, en dus de Arduino met een USB kabeltje aan de PC hebt hangen, en de Arduino IDE (software) hebt draaien. Uiteraard ga ik er dus ook van uit dat je het test programma, het knipperende LEDje (lampje), ook inderdaad aan de gang hebt gekregen.

Omdat de Arduino IDE het belangrijkste stukje gereedschap is voor het Arduino Programmeren, gaan we hier het programma een beetje beter bekijken.

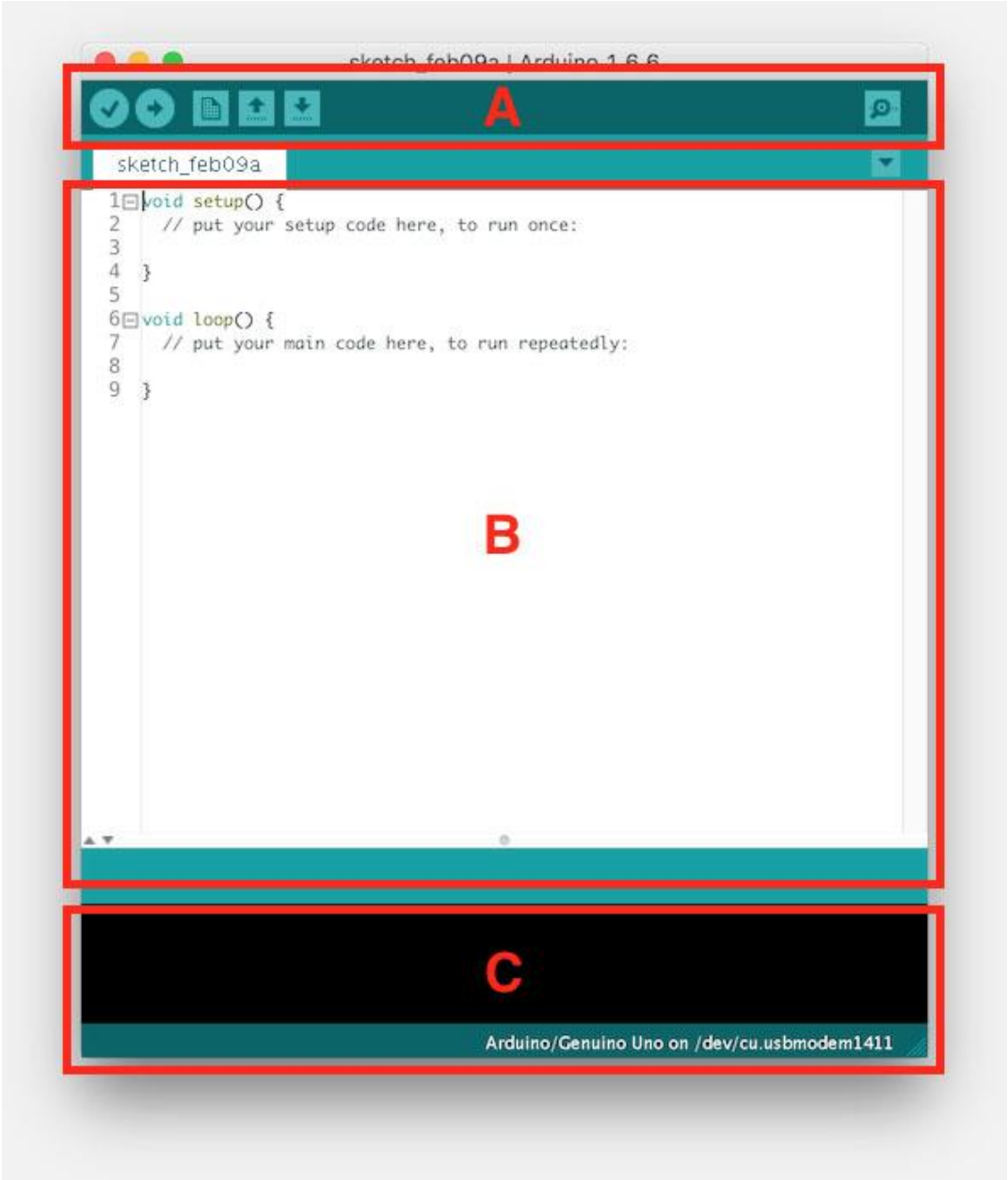
Laten we eerst even de Arduino IDE openen. Als dit niet de eerste keer is dat je de Arduino IDE start, klik dan op de knop "**C**", zoals in onderstaande afbeelding weergegeven, om een nieuw "project" te starten:



Figuur 2 - 1

Arduino Software – Handige snelkoppelingen

Na het klikken op de "C" knop, of het voor het eerst openen van de Arduino IDE, zul je een venster zien zoals hieronder:



Figuur 2 - 2

Arduino IDE – Belangrijke secties

We zien 3 “belangrijke” secties (zie afbeelding 2, hierboven):

In sectie “**A**” zien we de **knoppen** die we eerder besproken hebben (zie afbeelding 2-1) maar we zien nu ook nog helemaal rechts een extra knop die op een vergrootglas lijkt. Dit is de “**Serieel Monitor**” knop, welke soms ook weleens “Debug Window” knop genoemd wordt.

Figuur 2 - 3

Arduino IDE – Serieel Monitor Knop

Sectie "**B**" is waar we onze **Source Code (Broncode)** kunnen bewerken. Het is eigenlijk een simpele tekstverwerker met zogenaamde "code highlighting". *Code highlighting* zorgt ervoor dat bepaalde trefwoorden, tekens of character reeksen een bepaalde kleur krijgen, wat ons beter helpt bij het lezen van onze broncode.

Sectie "**C**" is waar the area de Arduino IDE ons belangrijke **meldingen** zal geven. Bijvoorbeeld als er iets fout gaat als we een onze broncode compileren, of als er iets anders fout of goed gaat. In het begin zullen de meldingen hier wel eens knap onduidelijk zijn, maar naarmate we meer leren en ervaring opdoen, worden ze gemakkelijker om te volgen.

De Arduino IDE heeft natuurlijk nog veel meer functies, die we zelden gebruiken maar toch handig zijn. Je vindt ze in de menu's en we hebben er al 2 gezien bij het aansluiten van de Arduino – toen we communicatiepoort en type Arduino moesten instellen.

Naarmate we meer werken met de Arduino, zul je merken wat een aantal van die functies voor ons kunnen doen, aar in het begin zijn ze niet zo van belang.

2.2. Communicatie met de Arduino …

Zoals eerder gezegd en gezien: de Arduino heeft geen beeldscherm, toetsenbord of muis. Dat maakt communicatie tussen ons mensen en de Arduino wat lastig en aan ee knipperend lampje hebben we niet al te veel. Dus hoe kunnen we zien (uitvoer!) wat de Arduino nou aan het sjouwen is?

Dit is nu waar de "**Serieel Monitor**" van pas gaat komen!

Wat USB te bieden heeft bij Arduino Programmeren

Laten we even kort kijken wat de USB-aansluiting te bieden heeft als we met een Arduino aan de slag gaan.

1. USB voorziet de Arduino van stroom.

Zoals met de meeste elektronica, heeft de Arduino natuurlijk stroom nodig om te werken. Onder normale omstandigheden biedt een USB-aansluiting 5V, wat genoeg is voor de Arduino om te werken.
Mochten we geen USB-aansluiting gebruiken dan zullen we een adaptertje moeten gebruiken om de Arduino van stroom te voorzien (ronde connector bij de meeste Arduino modellen).

2. USB als middel om programma's naar de Arduino te sturen

We schrijven ons programma op een computer, en dat programma (na vertaling) moet dus naar de Arduino overgezet worden – het zogenaamde "uploaden". Dit gebeurt ook over de USB-verbinding.

3. USB kan data ontvangen van de Arduino

De USB-verbinding kan echter ook gebruikt worden om gegevens (data) van de Arduino naar de PC terug te sturen, iets wat we gaan gebruiken als vervanger van het ontbrekende beeldscherm. Omdat we veel van de programmeertaal gaan uitproberen, gaat dit erg handig zijn.

Seriële Communicatie over USB

Zoals je ziet: de belangrijkste taak van de USB-verbinding is communicatie.

Nu moet je weten dat vroeger een Micro-Controller, zoals we die ook op de Arduino hebben, communiceerde via een zogenaamde seriële poort. Dat doen ze nog steeds, maar de handige mannen van het Arduino team hebben een "omzetter" op de meeste Arduino modellen gezet die dit omzet naar een USB-aansluiting. Een dergelijke seriële poort werd ook wel "**com poort**" of "**Serieel poort**" genoemd.

Een seriële-port, of com-poort, verstuurd gegevens (data) met een bit per keer – achter elkaar dus.
Een **bit** is iets wat één (1) of een nul (0) zijn.

Aan een enkele bit hebben we niet zo erg veel – het kan alleen maar 2 standen aangeven. Bijvoorbeeld AAN/UIT, JA/NEE, WAAR/NIET-WAAR (dit heet overigens "boolean", iets wat we later bespreken). Slimme mensen hebben daarom de "**byte**" bedacht.
Een "byte" combineert 8 bits in een groep, welke 256 verschillende waarden kan aannemen, waardoor het ineens wel zinvol wordt. We kunnen met 256 waarden bijvoorbeeld het hele alfabet (kleine letters en hoofdletters), nummers, en tekens omvatten.

Dus een bit kan 1 van 2 waarden aannemen, een nul (0) of een één (1).
Een byte heeft 8 bits, wat wiskundig wil zeggen $\mathbf{2^8}$ = 2 tot de macht 8 = 2 × 2 × 2 × 2 × 2 × 2 × 2 × 2 = 256 waarden. Mocht je op school nog niet gehoord hebben van "tot de macht", geen probleem … ik probeer het in een van de volgende delen een beetje uit te leggen, maar voor nu is het belangrijk dat je weet dat het 256 waarden kan hebben; van 0 … 255.

*"**Tot de macht**" geeft aan hoeveel keer een nummer gebruikt wordt voor vermenigvuldiging.*

Met andere woorden:
Hoe veel keer je het nummer "1" met dit nummer moet vermenigvuldigen.

Een seriële poort "praat" met een bepaalde snelheid, dit noemt men vaak de zogenaamde **baudrate**. De baudrate geeft aan hoe snel bits verstuurd en/of ontvangen worden, dus eigenlijk "bits per seconde". Er zijn een aantal standaard snelheden zoals bijvoorbeeld 9600, wat wil zeggen dat gegevens (data) met een snelheid van 9600 bits per second verstuurd worden, wat omgezet naar bytes wil zeggen 1200 bytes (of karakters) per seconde. Vergeet niet dat een byte uit 8 bits bestaat, dus 9600 : 8 = 1200.

Een kleine opmerking: om helemaal correct te zijn is de berekening van snelheid in bytes een beetje complexer dan wat ik hier net aangaf. Dit heeft o.a. te maken met controle of alles goed is aangekomen, indicatie wanneer een byte “klaar is”, etc. Maar … daar hoeven we ons nog niet zo druk over te maken.

Uiteraard moet men de snelheid aan beide kanten hetzelfde hebben – dus zowel de computer als de Arduino moeten met dezelfde snelheid praten anders snappen ze elkaar niet.

Een aantal standaard baudrate waarden zijn:

Standard Baudrates

Baudrate
110
300
600
1200
2400
4800
9600
14400
19200
38400
56000
115200

Er zijn nog meer standaardwaarden, en deze gaan vaak hoger dan 115200. Helaas wil het zo zijn dat niet iedere computer (met name Windows) daar altijd even goed in is, waar een Mac of een Linux computer hier altijd wel goed mee overweg kan.

OK, even genoeg geschiedenisles voor vandaag ….

Omdat de meeste computers tegenwoordig niet meer geleverd worden met een antieke seriële-poort of com-poort, gebruikt de Arduino een **USB-naar-Serieel converter**, welke op de meeste Arduino ’s te vinden zijn. Deze worden dan door de computer als een seriële-poort gezien.

We weten dus nu dat via de USB-verbinding ons programma naar Arduino gestuurd kan worden, tijd om eens te kijken hoe de Arduino nu gegevens terug kan sturen naar de computer.

Seriële Monitor – Gegevens van de Arduino ontvangen

Als we een programma starten, dan willen we natuurlijk graag zien wat het programma doet. Al is het maar voor onze voorbeelden of voor het zoeken naar fouten in ons programma – dat zoeken van fouten heet "**debugging**".

***Debugging** is het proces waarbij men fouten ("**bugs**") probeert te vinden en probeert te verhelpen in programma's.*

Hier is de "**Seriële Monitor**" erg handig …

Tip: Herstart een Arduino Programma met de Seriële Monitor

Elke keer als je de Seriële Monitor opent, krijgt de Arduino een "herstart" (reset) signaal waardoor het opnieuw opstart alsof je de Arduino net hebt aangezet, zodat het programma netjes schoon opstart.

Je kunt dit dus gebruiken als truc om de Arduino opnieuw op te starten: Sluit de Seriële Monitor en open de Seriële monitor weer opnieuw en de Arduino start weer opnieuw.

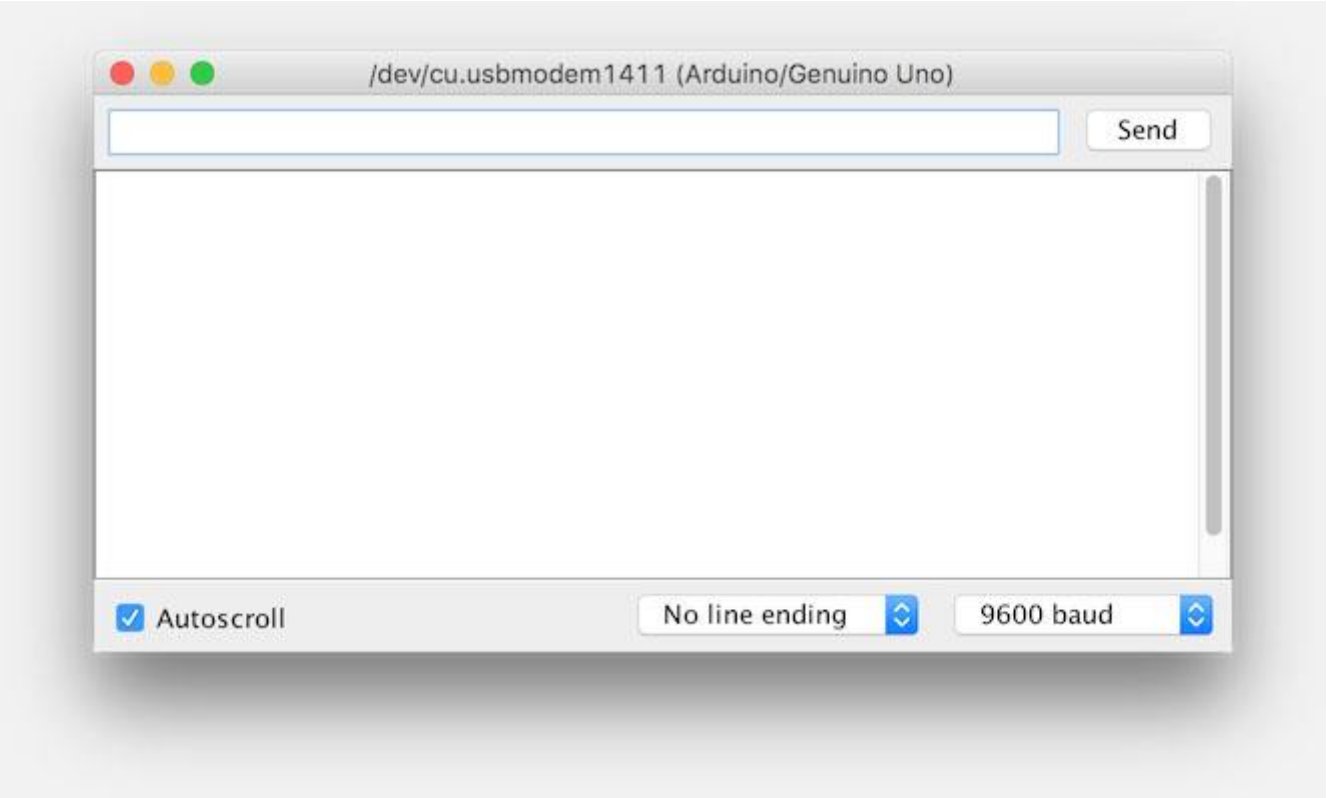
Om er nu voor te zorgen dat de Arduino ons informatie stuurt, moeten er dus voor zorgen dat zowel computer als Arduino op dezelfde snelheid babbelen. We moeten zowel de Seriële Monitor als de Arduino dus vertellen met welk tempo we gaan communiceren.

Voor de Arduino wil dat zeggen dat we in ons programma moeten opgeven welke baudrate we willen gaan gebruiken. Meestal is de Arduino IDE slim genoeg om dat vanzelf te ontdekken, maar soms moeten we het handmatig even doorgeven.

Laten we er even een voorbeeld bij gaan halen. Uiteraard moeten we hiervoor de **Arduino IDE open hebben**, en de **Arduino verbonden** met de computer, en natuurlijk op "**Seriële Monitor**" knop (het vergrootglas-knopje rechtsboven in de Arduino IDE).

Arduino IDE – Serieel Monitor Knop

Nadat je op de "Seriële Monitor" knop hebt gedrukt, krijg je of een foutmelding in de berichten sectie (Arduino niet correct aangesloten?) of een venstertje zoals hieronder:



Figuur 2 - 4

Arduino Seriële Monitor venster

In dit venster kunnen we gegevens (data) ontvangen van de Arduino, maar (later meer hierover) ook naar Arduino toe sturen.

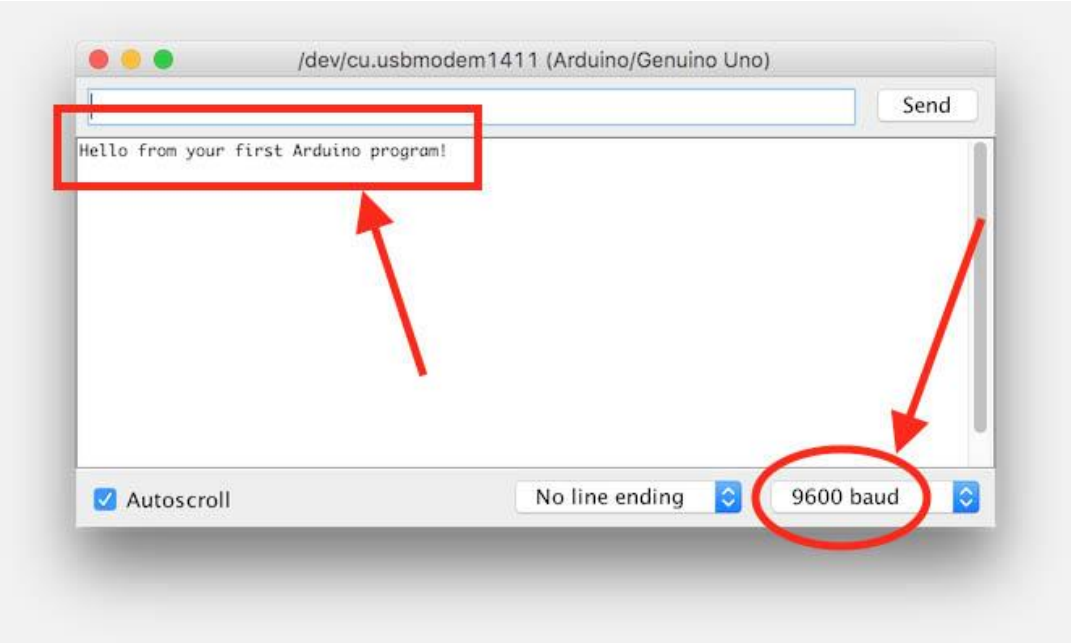
Laten we even ons eerste “praat” programma bekijken …
In de onderstaande code heb ik 2 regels aan de standaard code toegevoegd: regels 3 en 4.

```
1  void setup() {
2    // put your setup code here, to run once:
3    Serial.begin(9600);
4    Serial.println("Hello from your first Arduino program!");
5  }
6
7  void loop() {
8    // put your main code here, to run repeatedly:
9
10 }
```

De regel “`Serial.begin(9600);` ” vertelt de Arduino dat het over de seriële verbinding moet praten met een snelheid van 9600 baud.
Als je nu in de volgende afbeelding kijkt, dan zie je “9600 baud” rechts onderin – dezelfde snelheid als wat we in het programma opgeven.

De daarop volgende regel “`Serial.println("Hello from your first Arduino program!");` ” (Engels voor: Hallo van jouw eerste Arduino Programma) vertelt de Arduino dat het tekst moet versturen over de seriële verbinding.

OK, kopieer nu de code en plak het in de Arduino IDE, waarbij alle bestaande tekst vervangen wordt.
Klik vervolgens op de “Compileer en Upload” knop (knop “B” in afbeelding 1).
De Arduino zal weer wat knipperen met de LEDjes, als dat afgelopen is dan zul je het volgende in het venster van de Seriële Monitor zien:



Figuur 2 - 5

Onze eerste Arduino Output

Ziet er toch best eenvoudig uit hè? Als je het eenmaal gezien hebt althans …

2.3. De Basis Arduino Programma Lay-out

Setup() en Loop()

Ieder Arduino programma heeft dezelfde basis lay-out en dat zie je meteen als je een nieuw project start (klik op de "C" knop in afbeelding 1).

```
1  void setup() {
2    // put your setup code here, to run once:
3  }
4
5  void loop() {
6    // put your main code here, to run repeatedly:
7
8  }
```

Hier zien we twee basis elementen: "**setup()**" en "**loop()**", en beiden, ook al laat je ze leeg, moeten bestaan!

Als de Arduino namelijk start, dan zal het eerst naar "setup()" gaan zoeken en deze slechts èèn keer doorlopen.
Daarna zal de Arduino "loop()" uitvoeren en deze "loop()" oneindig blijven herhalen. Vandaar dus de naam "loop" (lus).

De Arduino makers wisten dat natuurlijk al, dus vandaar dat de Arduino IDE deze code al automatisch gaat aanmaken.

Elk Arduino programma heeft een "setup()" functie welke tijdens het opstarten slechts één maal de startup() doorloopt, en vervolgens de "loop()" functie eindeloos blijft herhalen tot de Arduino uitgezet wordt.

Om dit te illustreren zouden we de code die de tekst uitvoert naar de seriële verbinding, in de "loop()" kunnen zetten in plaats van in de "setup()". Het gevolg zou zijn dat het venster van de Seriële Monitor herhaaldelijk dezelfde tekst blijft weergeven tot we zeg maar de stroom van de Arduino af halen (USB kabel er uit trekken).

We laten "`Serial.begin(9600);`" in de "setup()" staan, want instellen van de snelheid hoeft natuurlijk maar 1 keer.
De regel "`Serial.println("Hello from your first Arduino program!");`" verplaatsen we nu naar de "loop()" wat dus als gevolg heeft dat die zich blijft herhalen.

```
1  void setup() {
2    // put your setup code here, to run once:
3    Serial.begin(9600);
4  }
5
6  void loop() {
7    // put your main code here, to run repeatedly:
8    Serial.println("Hello from your first Arduino program!");
9  }
```

2.4. Arduino Programmeren met Opmerkingen

Het toevoegen van opmerkingen in de code van ons programma is zeker sterk aan te raden. Als eerst maken de opmerkingen onze code aanzienlijk duidelijker, wat handig is als we op een later tijdstip iets willen veranderen in onze code.
Nog belangrijker is dat opmerkingen uitleg bieden aan anderen die misschien jouw code willen bewerken of aanpassen.

Plaats altijd Opmerkingen (uitleg) in jouw code!

In een aantal van de eerdere code voorbeelden heb je al wat opmerkingen in de code zien staan, het zijn de regels die met 2 schuine strepen, ook wel slashes ("**//**") genoemd, beginnen.

Maar er zijn meer opties om opmerkingen te plaatsen.

Als eerste weten we dat opmerkingen beginnen met 2 schuine strepen "**//**", wat wil zeggen dat alles achter deze tekens (in deze regel) als commentaar of opmerking gezien moet worden. Dit wil dus ook zeggen dat de vertaler (de compiler) alles na de 2 schuine strepen negeert op betreffende regel.
Dit wil ook zeggen dat een opmerking achter commando's kan staan.

Opmerkingen maken een programma niet groter of trager,

Opmerkingen maken alleen de Source Code groter, maar ook meteen aanzienlijk beter leesbaar en duidelijker.

Opmerkingen kunnen ook tussen "**/***" en "***/**" geplaatst worden en dat mag zelfs over meerdere regels. Een aantal voorbeelden:

```
1  // Enkele regel opmerking
2  Serial.begin(9600);  // Opmerking na een commando
3
4  // De volgende regel schakelt tijdelijk een commando uit
5  // Serial.begin(9600);
6
7  /* Dit is een opmerking blok die slechts 1 regel beslaat */
8
9  /* Maar een opmerking blok
10 kan ook meerdere
11 regels beslaan      */
12
13    /* Met een opmerking blok kun je ook meerdere regels met commando's tijdelijk 15
14 uitschakelen
15 Serial.begin(9600);
16 Serial.println("Hello from your first Arduino program!");
17 */
```

Hier zien we dus opmerking als enkele regel, aan het einde van een regel, of meerdere regels beslaan.

Wat we ook zien is dat het ook handig gebruikt kan worden om bepaalde stukken van onze code uit te schakelen – bijvoorbeeld als we wat aan het testen zijn tijdens de ontwikkeling van ons programma.

2.5. Statements en Code blokken

Misschien heb je het gemerkt, misschien niet, maar ik zat net even te klungelen met het juiste woord voor "commando's" want een echt goede vertaling in het Nederlands voor "statement" kon ik zo 1-2-3 niet vinden. Elke regel kan namelijk leeg zijn, een opmerking of een instructie (**statement**) bevatten. Een instructie, of statement, wordt altijd afgesloten

met een puntkomma (**;**). Dit moet je goed onthouden want dit kan een van de meest voorkomende type fouten zijn in een programma.

*Een instructie of **statement eindigt altijd met een puntkomma** (;) …*

Het woord "instructie" is beter Nederlands, maar in de computerwereld gebruikt men vaak de Engelse taal en zeggen we dus vaker "statement".

Code blokken is weer zo'n leuke term en die hebben we al een paar keer gezien. In eerder

```
1  void setup() {
2    // put your setup code here, to run once:
3    Serial.begin(9600);
4  }
```

Voorbeelden zagen we bijvoorbeeld 2 van dit soort "code blokken" ingesloten met accolades.

Als we dit voorbeeld eens goed gaan bekijken dan zien we in de eerste regel **"void setup() {**". Dit wil zeggen dat we hier een functie definiëren met de naam "setup", welke geen waarde terug stuurt (void) en geen parameters neemt (de twee ronde haakjes) – dit gaat al verder dan wat we behandeld hebben en in het hoofdstuk "functies" gaan we hier dieper op in. Voor nu onthouden we:
"*Hier start de definitie van de functie setup, welke geen parameters gebruikt en niets terug geeft*".

De twee ronde haakjes **()**, geven aan wat voor informatie we naar de functie willen sturen – en omdat er niets staat: niks dus!
De term "**void**" is zoiets als luchtledig, niks, noppes, nada, niets, … en omdat het voor de functie naam "setup" staat wil dit zeggen dat de functie geen informatie terug zal geven. Je hoeft dit nog niet te onthouden – zoals gezegd; we behandelen de details later als we naar functies gaan kijken.

*Een code blok start en eindigt met accolades.(**{ … }**)..*

De accolade open, start de zogenaamde "code blok". Alles wat hierna geschreven wordt, tot het punt waar we accolade sluiten tegen komen, wordt als 1 enkele blok gezien. Dus in het voorbeeld de code die bij de functie setup hoort "setup". De opmerking en de regel met "Serial.begin(9600);" horen dus bij elkaar, in een blok, welke bij de functie "setup" hoort.

Nu hebben we wat meer basisinformatie, dus tijd om eens te gaan kijken naar wat interessantere programmeer zaken ….

Als je vragen hebt: stel ze dan hieronder, en bedenk dat er geen domme vragen zijn, behalve dan natuurlijk de vraag die niet gesteld is. We zijn allemaal een keer bij nul begonnen!

Volgende hoofdstuk: **Arduino Programmeren voor Beginners – Deel 3: Werken met Informatie**