Structuur Arduino

De basisstructuur van de Arduino programmeertaal is erg simpel. Het bestaat uit minstens twee gedeeltes (blokken). Deze twee gedeeltes (blokken), of functies vormen een aantal statements (programma commando's). Voorbeeld:

```
void setup()
{
statements;
}
void loop()
{
statements;
}
```

Waarbij setup() de voorbereiding is en loop() de uitvoering. Beide functies zijn nodig
om een programma te kunnen laten werken.
In de setup functie worden variabelen gedeclareerd en bepaald welke pinnen ingang of uitgang worden. De setup functie wordt slechts eenmaal doorlopen. De loop functie volgt na de setup functie en wordt vaak oneindig herhaalt. De loop
functie leest vaak de inputs en laat afhankelijk daarvan bepalen wat de outputs moeten doen. Eigenlijk komt het er op neer dat de loop functie de motor van het
programma is dus daar waar al het werk moet gebeuren.

setup()

De setup() functie wordt één keer aangeroepen wanneer het programma start. Het wordt gebruikt om pin modes te initialiseren of begint met seriële communicatie.
De syntax ziet er als volgt uit:.

```
void setup()
{
pinMode(pin, OUTPUT); // maak de 'pin' als uitgang
}
```

loop()

Nadat de setup() functie aangeroepen is volgt de loop() functie. Deze doet precies
wat de naam zegt en loopt constant te meten om vervolgens te reageren door veranderingen aan te brengen. De loop functie is een oneindige loop. De syntax:

```
{
digitalWrite(pin, HIGH); // zet 'pin' aan
```

```
delay(1000); // Eén seconde pauze
digitalWrite(pin, LOW); // zet 'pin' uit
delay(1000); // Eén seconde pauze
}
```

functies

Een functie is een blok met code dat een naam heeft gekregen en waarin instructies staan die uitgevoerd moeten worden wanneer de functie aangeroepen wordt. De functies void setup() en void loop() zijn al genoemd, maar andere functies kunnen ook. Er zijn ook op maat gemaakte functies die de het aantal opdrachten in een
programma reduceren, waardoor het geheel overzichtelijker is.
Functies moeten eerst gedeclareerd worden in hun type.
Dat ziet er als volgt uit:

```
type functionName(parameters)
{
statements;
}
```

Een voorbeeld: De volgende type integer functie delayVal() wordt gebruikt om een vertraging in te bouwen bij het uitlezen van een waarde van een aangesloten potentiometer. Als eerste wordt een lokale variabele v gedeclareerd die vervolgens
een waarde krijgt tussen 0 en 1023. In een volgende regel is een functie v /=4; Hier wordt v door 4 gedeeld omdat de maximale waarde in een byte 255 kan zijn. Met de return opdracht gaat het programma terug naar zijn hoofdprogramma.

```
int delayVal()
{
int v; // maak een tijdelijke variabele 'v'
v = analogRead(pot); // lees de potentiometer waarde
v /= 4; // converter 0-1023 naar 0-255
return v; // return definitieve waarde
}
```

{ } krullende haakjes (accolade)
Krullende haakjes geven het begin of het einde aan van een functieblok zoals je ook tegenkomt bij de void loop(). Kijk maar:

```
type functie()
{
statements;
}
```

Het eerste opende accolade (gekrulde haakje) { moet altijd afgesloten worden door een (gekruld gesloten) accolade }. Het aantal accolades is dus altijd een even getal. Let daar goed op want één accolade te weinig en een heel programma kan stoppen met werken. Vind dan de ontbrekende accolade maar eens terug.

; puntkomma

Een puntkomma moet gebruikt worden na elke ingevoerde opdracht. Net zoals de gekrulde haakjes zal bij het ontbreken van een puntkomma een error verschijnen.

Voorbeeld:

```
int x = 13; // declareer variabele 'x' als een integer 13
```

Opmerking: Vaak is het zo dat het ontbreken van een puntkomma er voor zorgt dat de Arduino software niet wil compileren en een error aangeeft op een andere plek dan waar de puntkomma vergeten is. Dat wordt dus lastig zoeken.

/*… */ blok commentaar

Blok commentaar zijn gebieden met tekst die door het programma genegeerd worden. Tussen de /* en */ staat meestal uitleg over het programma of over de code die daar staat. Het mooie van “blok commentaar” is dat het over meerdere regels
geplaatst kan worden.

/* dit is een blok met commentaar
Vergeet niet het einde van het
Commentaar aan te geven
*/
Commentaar wordt nooit mee geprogrammeerd in de microcontroller. Het neemt
dus geen geheugenruimte in beslag. Natuurlijk wordt het wel bewaard in het Arduino programma (Windows/Linux/Mac).
// regel commentaar
Een regel die begint met // en eindigt met tekst of code zal op die regel genegeerd
worden. Ook dit neemt geen geheugen in de microcontroller in beslag. Code voor
in de regel gevolgd door // zal wel uitgevoerd worden alles wat na de // komt op die regel echter niet.
// Dit is commentaar op een regel en onderstaande wordt
// niet uitgevoerd omdat het vooraf gaat met //.
// A = B + 3
Regel commentaar wordt vaak gebruikt om de genoemde opdrachten uit te leggen.
Opmerking: Als een programma niet werkt zoals het zou moeten werken dan is het
vaak handig om een gedeelte van het programma uit te schakelen door stukken van je programma te voorzien van //. Je kunt dan stapsgewijs onderzoeken waar de
fout zit.

Variabelen

Een variabele is een manier om een numerieke waarde te bewaren voor later gebruik in het programma. Zoals de naam variabele al aangeeft kan de waarde van een variabele ook regelmatig veranderen. Er bestaan ook zogenaamde constanten. Dat zijn variabelen die constant het zelfde blijven en dus nooit van waarde veranderen. Een variabele moet op een juiste manier gedeclareerd worden. In de code hier onder wordt een variabele gedeclareerd genaamd inputVariabele die vervolgens de waarde krijgt die gemeten wordt op de analoge
input pin 2:
int inputVariabele = 0; // declareer een variabele en geef
// die de waarde 0

```
inputVariabele = analogRead(2); // geef de variabele de waarde die
// gelezen wordt op de analoge pin 2
```

‘inputVariabele’ is de variabele zelf.
Op de eerste regel staat dat er een variabele
gedeclareerd wordt van het type int (int is de afkorting voor integer).
De tweede regel krijgt de variabele de waarde toegewezen die gemeten wordt op de analoge pin 2. Later in het programma zal er wel wat met die variabele gedaan worden. Vaak zal de variabele getest worden of hij aan bepaalde condities voldoet.
Een voorbeeld: De volgende code test of de inputVariabele kleiner is dan 100. Als dat zo dan krijgt inputVariabele de waarde 0. Is de waarde hoger dan 100 dan houdt inputVariabele de waarde die hij al had. In de derde regel zie je dat een pauze gemaakt is die dus alleen maar optreedt als inputVariabele groter of gelijk is aan 100.

```
if (inputVariabele < 100) // test of de variabele kleiner
// is dan 100
{
inputVariabele = 0; // zo ja, dan krijgt hij de waarde 0
}
delay(inputVariabele); // De waarde van de variabele bepaalt
// de pauze
```

Opmerking: Geef variabelen een logische naam bijvoorbeeld tiltSensor of pushButton. Bekijken anderen jouw programma dan wordt het al een stuk makkelijker lezen. Je kunt elk woord voor variabelen gebruiken tenminste als het geen naam is die al gebruikt wordt in de Arduino omgeving.

Variabelen declareren

Alle variabelen moeten eenmalig gedeclareerd worden voordat je ze kunt gebruiken. Er zijn verschillende types zoals int, long, float, etc.

Variable bereik

Een variabele kan gedeclareerd worden in het begin van het programma voor void
setup(). Soms heb je door omstandigheden een variabele in een programma niet nodig. Daarom kun je ook een variabele later in het programma wel of niet aanmaken al naar gelang hij nodig is.
De vaste variabele heet een globale variabele. Een globale variabele is dus een variabele die in een heel programma kunt oproepen. Deze variabele declareer je boven void setup().
Een locale variabele is een variabele die alleen gebruikt kan worden in een stukje

van een programma. Het is een tijdelijke variabele. Zo'n stukje kan bijvoorbeeld in
de void loop() zitten. De reden dat deze variabelen bestaan is dat de tijdelijk geheugen in beslag nemen en daardoor efficiënter met het geheugen van de microcontroller wordt omgesprongen.
Opmerking: Hoe meer variabelen je gebruikt in een microcontroller des te sneller zal
het geheugen vol zitten. Dat kan natuurlijk niet de bedoeling zijn.
Het volgende voorbeeld zal duidelijk maken hoe de verschillende variabelen werken:

```
int value; // 'value' is zichtbaar in het hele
// programma
void setup()
{
// geen setup nodig
}
void loop()
{
for (int i=0; i<20;) // 'i' is alleen zichtbaar in de for loop
{
i++; // i = i + 1
}
float f; // 'f' is alleen zichtbaar in de inside
// loop
```

Op de volgende bladzijde worden de verschillende type variabelen beschreven.

byte

Byte bewaart een 8-bit numerieke waarde zonder een decimale punt met een bereik van 0-255.

```
byte Button1 = 180; // declareert 'Button1' als een byte type
```

int

Integers zijn primaire datatypes om getallen te bewaren zonder een decimale punt een 16-bit waarde met een bereik van 32767 tot -32768.

```
int Count4 = 1500; // declareert 'Count4' als een integer type
```

Opmerking: Een integer variabele kan niet groter zijn dan 32767. Verhoog je 32767 met 1 dan wordt het een negatief getal: -32768.

long

Datatype voor erg grote getallen, zonder een decimale punt (een soort uitgebreide integer) een 32-bit waarde met een bereik van 2,147,483,647 tot -2,147,483,648.

```
long someVariabele = 90000; // declareert 'someVariabele'
// als een long type
```

float

Een datatype voor getallen met een decimale punt. Dus met getallen achter de komma. Floating datatypes nemen meer geheugen in gebruik dan een integer en worden opgeslagen als een 32-bit waarde met een bereik van 3.4028235E+38 tot -3.4028235E+38.

```
float someVariabele = 3.14; // declareert 'someVariabele' als
// een float-point type
```

Opmerking: Floating-point getallen zijn of hoeven in principe niet gelijk te zijn als je ze met elkaar vergelijkt. Met het rekenen aan floating getallen heeft de microcontroller veel meer tijd nodig dan bij byte of integer getallen.

De Arduino microcontroller Pagina 12

Arrays

Een array is a verzameling van verschillende waardes die benaderd kunnen worden door een indexnummer. Je kunt er elke waarde in kwijt en je kunt het oproepen door de naam van de variabele en de indexnummers. Arrays zijn standaard met nullen gevuld en het eerste indexnummer van een array begint ook met een 0.

int myArray[] = {waarde0, waarde1, waarde2...}

Het is mogelijk om een array te declareren naar type en grootte en dan later de waardes in te vullen op basis van een index-positie:

int myArray[5]; // declareert integer array met 6 positions

myArray[3] = 10; // vul de 4e index positie met de waarde 10

Om de waarde terug te krijgen:

x = myArray[3]; // x is nu gelijk aan 10

Arrays worden vaak gebruikt in loops waarin tellers zitten die telkens met 1 opgehoogd worden zodat ze makkelijk traceerbaar zijn. Het volgende voorbeeld gebruikt een array om een LED te laten knipperen. Er wordt een loop gebruikt. De teller begint op 0, schrijft die waarde op de index positie in de array flicker[], in dit geval 180 op de PWM pin 10, wacht 200 ms en gaat vervolgens naar de volgende index positie.

```
int ledPin = 10; // LED on pin 10
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60};// array van 8
void setup() // different values
{
pinMode(ledPin, OUTPUT); // vul de OUTPUT pin
}
void loop()
{
for(int i=0; i<7; i++) // loop gelijke nummers
{ // of values in array
analogWrite(ledPin, flicker[i]); // schrijf index waarde
delay(200); // pause 200ms
}
}
```

Opmerking: Voor de werking van PWM staat een voorbeeld in de bijlage. Je kunt door pulsen te geven de indruk wekken dat een LED feller of minder fel licht geeft.

De Arduino microcontroller Pagina 13

Rekenen

Rekenkundige bewerkingen zijn bijvoorbeeld optellen, aftrekken, vermenigvuldigen en delen. Het is altijd een resultaat van twee getallen (operands). Voorbeelden:

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

De berekening wordt uitgevoerd afhankelijk van het gekozen datatype. Als er gekozen is voor een integer dan zal het resultaat 9 / 4 = 2 zijn in plaats van 2.25. Pas ook op bij rekekundige bewerkingen dat er een “overflow” error kan komen bij te grote getallen. Een byte kan tot maximaal 255, zodat een variabele die gedeclareerd is als een byte de optelling 250 + 23 niet kan onthouden.

Zijn de getallen die je gaat bewerken van twee verschillende types, dan wordt het grootste type gebruikt voor de berekening. Bijvoorbeeld ans één van de getallen een integer is en het andere getal een float dan zal de uitkomst een getal zijn van het type float.
Kies dus altijd een type variabele die groot genoeg is voor de gewenste berekening. Zorg dat je weet wat er gebeurt als de gebruikte variabele door een optelling ineens van positief veranderd in negatief. Weet je het niet zeker lees dan een paar pagina's terug hoe je getallen moet declareren. Let echter wel op dat float variabelen veel geheugen in beslag nemen en ook de microcontroller zwaarder belasten (lees: langzamer werken).

Samengestelde opdrachten

Samengestelde opdrachten voor simpele wiskundige berekeningen kent de Arduino ook. Ze worden veel gebruikt in loops en worden later nog beschreven in deze manual. De meest voorkomende samengestelde opdrachten zijn:
x ++ // is hetzelfde als x = x + 1, of verhoog x met +1
x -- // is hetzelfde als as x = x - 1, of verlaag x met -1
x += y // is hetzelfde als as x = x + y, of verhoog x met +y
x -= y // is hetzelfde als x = x - y, or of verlaag x met -y
x *= y // is hetzelfde als x = x * y, of vermenigvuldig x met y
x /= y // is hetzelfde als x = x / y, of deel x met y

Vergelijken van getallen

Variabelen worden vaak met elkaar vergeleken. Op basis daarvan worden er dan beslissingen genomen. Op deze en de volgende pagina's staan daar veel voorbeelden van.
x == y // x is gelijk aan y
x != y // x is niet gelijk aan y
x < y // x is kleiner dan y
x > y // x is groter dan y
x <= y // x is kleiner of gelijk aan y
x >= y // x is groter of gelijk aan y

Logische berekeningen

Logische berekeningen zijn vergelijkingen die als uitkomst hebben waar of niet waar (TRUE of FALSE). Er zijn drie logische operaties, AND, OR, en NOT die vaak gebruikt worden in zogenaamde if opdrachten:
Logische AND:
if (x > 0 && x < 5) // waar alleen als beide
// vergelijkingen waar zijn
Logische OR:
if (x > 0 || y > 0) // waar als één van de twee
// vergelijkingen waar zijn
Logische NOT:
if (!x > 0) // alleen waar als
// vergelijking niet waar is

Constantes

De Arduino taal heeft een aantal voorgedefinieerde waardes, die ook wel constantes worden genoemd. Ze worden gebruikt om een programma makkelijker te kunnen lezen of schrijven. Constantes zitten ook in verschillende groepen.
true/false

Dit zijn Boolean constantes die een logisch niveau vaststellen. False wordt gedefinieerd als een 0, terwijl true wordt gedefinieerd als een 1 of iedere andere waarde anders dan een 0. In Boolean is -1, 2 en -900 gedefinieerd als true.
Voorbeeld:

```
if (b == TRUE);
{
Doe iets;
}
```

high/low

Deze constantes definiëren een pin niveau van HIGH of LOW en worden gebuikt om digitale pennen te lezen. HIGH is een logische 1 en LOW is een logische 0. Een logische 1 is meestal 5 V, maar er bestaat ook al een Arduino waarbij dat 3,3 V is.
Voorbeeld:

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

input/output

Deze constantes worden gebruikt met de opdracht pinMode() om te definiëren of een digitale pin INPUT of OUTPUT moet worden. Voorbeeld:

```
pinMode(13, OUTPUT); \\Pin 13 is een output
```


if

De if opdracht test of bepaalde condities bereikt zijn. Denk bijvoorbeeld aan een analoog signaal dat een bepaalde waarde bereikt waarbij ingegrepen moet worden. In dat geval moet er iets gebeuren. Die actie moet dan plaats vinden binnen de haakjes (zie het voorbeeld hier onder). Wordt er niet aan de voorwaarde voldaan dan wordt de actie tussen de haakjes overgeslagen.
Voorbeeld:

```
if (waardeVariabele ?? waarde)
{
Doe iets;
}
```

In het bovenstaande voorbeeld wordt de waardeVariabele vergeleken met een andere waarde. Die waarde kan echter ook een constante zijn zoals genoemd op de vorige pagina.
Opmerking: Pas op met het volgende te gebruiken: if(x=10). Deze is technisch gezien juist. Het geeft x de waarde 10 en heeft als resultaat altijd TRUE. Gebruik liever '==' zodat bij de opdracht if(x==10) getest wordt gelijk is aan de waarde 10 of niet. Bedenk: bij '=' aan de term gelijk en bij '==' aan de term is gelijk aan.

if… else

De if… else opdracht maakt het mogelijk hoe dan ook een beslissing te laten nemen. Bijvoorbeeld je meet dat een digitale input pin hoog is in dat geval wil je dat actie_A start. Is de pin echter laag dan moet actie_B starten Dat zou er als volgt uit kunnen zien:.

```
if (inputPin == HIGH)
{
Voer aktie_A uit;
}
else
{
Voer aktie_B uit;
}
```

Else kan ook een andere procedure zijn zodat je meerdere testen in dezelfde lus kunt verwerken. Bekijk het volgende voorbeeld eens:

```
if (inputPin < 500)
{
Voer aktie_A uit;
}
else if (inputPin >= 1000)
{
Voer aktie_ B uit;
}
else
{
Voer aktie_C uit;
}
```

Opmerking: Kijk goed naar de haakjes en de puntkomma's dat wil op deze wijze best wel eens ingewikkeld worden.

for

Het for commando wordt gebruikt om een aantal commando's een bekend aantal keren te laten herhalen. Via een teller wordt bijgehouden hoe vaak de lus zich moet herhalen. Het commando ziet er als volgt uit:

```
for (variabele; conditie; expressie)
{
doeiets;
}
```

Dat lijkt lastig, maar het is een makkelijke en vaak gebruikte opdracht.
Een voorbeeld:

```
for (int i=0; i<20; i++) // declareer i, en test of
// het kleiner is
{ // dan 20, I wordt met 1 opgehoogd
digitalWrite(13, HIGH); // zet pin 13 aan
delay(250); // 1/4 second pauze
digitalWrite(13, LOW); // zet pin 13 uit
delay(250); // 1/4 second pauze
}
```

In de eerste regel wordt i gedeclareerd en wordt meteen getest of i kleiner is dan 20. Daarna wordt i met 1 verhoogd (en krijgt de waarde 1). Aangezien i 0 was wordt alle code die er onder staat tussen de haakjes uitgevoerd. Nadat dat is gedaan wordt regel 1 opnieuw uitgevoerd. i heeft nu de waarde 1 er wordt weer getest of i kleiner is dan 20, hetgeen nog steeds het geval is en i wordt met 1 verhoogd zodat i de waarde 2 krijgt. Dat blijft zich herhalen tot i de waarde 20 bereikt. Daarna wordt de code tussen de haakjes niet meer uitgevoerd.

Opmerking: In C is de for loop veel meer flexibeler in te vullen dan in sommige andere computer talen zoals bijvoorbeeld BASIC. De variabele, conditie en expressie kun je naar wens aanpassen. Let op: ze worden wel gescheiden door een puntkomma.

while

De while loop heeft wel wat weg van de for loop. Hij is gemakkelijk uit te leggen met: Zolang je aan die voorwaarde voldoet moet je dat doen. Die voorwaarde zou bijvoorbeeld het testen van een sensor kunnen zijn. De loop stopt pas als hij niet

meer aan de voorwaarde voldoet. Een voorbeeld:

```
while (someVariable ?? value)
{
doe iets;
}
```

Het volgende voorbeeld test of de someVariabele kleiner is dan 200. Als dat waar is blijft de lus zich herhalen totdat someVariabele niet langer kleiner is dan 200.

```
while (someVariabele < 200) // test of someVariabele kleiner
// is dan 200
{
Doe iets; // voer programmacode uit
someVariabele++; // verhoog variabele met 1
}
```

do… while

De do loop loop die grotendeels hetzelfde werkt als de while loop. Het verschil zit hem in het feit dat de conditie onderaan staat in plaats van bovenaan, zoals bij de while loop. Ongeacht de condities wordt de loop altijd 1 keer doorlopen.

```
do
{
doeiets;
} while (someVariable ?? value);
```

In het volgende voorbeeld wordt x hetzelfde als de waarde readSensors(), daarna volgt een pauze van 50 milliseconde, waarna de loop zich herhaalt totdat x is niet meer kleiner dan 100:

```
do
{
x = readSensors(); // x krijgt de waarde readSensors()
delay (50); // pauze 50 milliseconde
} while (x < 100); // herhaal totdat x is kleiner dan100
```


pinMode(pin, mode)

Wordt gebruikt in de void setup() om een specifieke pin te configureren als een INPUT of een OUTPUT.

```
pinMode(pin, OUTPUT); // sets ‘pin’ to output
```

Arduino’s digitale pinnen zijn standaard geconfigureerd als inputs. Je hoeft ze dus niet per sé te declareren als inputs met pinMode(). Pinnen die geconfigureerd zijn als INPUT bevinden zich in een hoogohmige toestand.

Er zitten ook hoogohmige weerstanden (pullup) van 20K ingebouwd in de Atmega chip die bereikt kunnen worden door de Arduino software. Dat kan op de volgende manier:

```
pinMode(pin, INPUT); // maak van ‘pin’ een input
digitalWrite(pin, HIGH); // schakel op de ‘pin’ de
//pullup weerstanden in
```

Pullup weerstanden worden normaal gesproken gebruikt bij met schakelaars die op de inputs aangesloten worden.

Pinnen die geconfigureerd zijn als OUTPUT staan in een laagohmige toestand en kunnen maximaal 40 mA leveren. Dit is ruim genoeg voor een LED, maar veel te weinig voor een motor of bijvoorbeeld een relais.

Kortsluiting tussen verschillende poorten kunnen de poort of de gehele Atmega chip onherstelbaar beschadigen. In dat geval moet de chip vervangen worden (inclusief een nieuwe bootloader).

Belangrijk: Het zou niet verkeerd zijn om outputs te beveiligen met een weerstand van 220Ω. Bij kortsluiting loopt er dan een stroom van I = U/R = 5/220 = 22 mA hetgeen kleiner is dan de eerder genoemde 40 mA.

De Arduino microcontroller Pagina 21

digitalRead(pin)

Leest de waarde uit van een specifieke pin met als resultaat HIGH of LOW. De pin is gespecificeerd al een variabele of een constante (0-13).

```
Value = digitalRead(Pin); // maak 'value' gelijk aan
// de input pin
```

digitalWrite(pin, value)

Schrijf de waarde naar een specifieke pin met als niveau HIGH of LOW. De pin is gespecificeerd als een variabele of een constante (0-13).

```
digitalWrite(pin, HIGH); // maak 'pin' hoog
```

Het volgende voorbeeld leest een drukknop uit die is aangesloten op pin 7 en laat een LED, aangesloten op pin 13 aan gaan als de drukknop ingedrukt is:

```
int led = 13; // LED op pin 13
int pin = 7; // drukknop op pin 7
int value = 0; // variabele value
void setup()
{
pinMode(led, OUTPUT); // maak van pin 13 een output
pinMode(pin, INPUT); // maak van pin 7 een input
}
void loop()
{
value = digitalRead(pin); // maak van 'value' de input pin
digitalWrite(led, value); // zet ‘value’ over naar de 'led'
}
```

De Arduino microcontroller Pagina 22

analogRead(pin)

Leest de waarde van een specifieke analoge pin in een 10 bit resolutie. Deze functie werkt alleen op pin 0 t/m 6 (Geldt natuurlijk niet voor de Arduino mega). De uitkomst is een integer waarde tussen 0 to 1023.

```
waarde = analogRead(pin); // maak van ‘waarde' wat gelezen
// wordt op de 'pin'
```

Opmerking: Analoge pinnen hoeven niet te worden gedeclareerd als INPUT of OUTPUT. Het zijn automatisch al digitale inputs.

analogWrite(pin, value)

Schrijft een pseudo-analoge waarde door gebruik te maken van hardwarematige puls-breedte (width) modulatie (PWM) naar een output pin die gemarkeerd is als PWM. Op nieuwere Arduinos met de ATmega168/368 chip, werkt deze functie op pin 3, 5, 6, 9, 10, and 11. Op oudere Arduinos met een ATmega8 werkt deze functie alleen op pin 9, 10, en 11. De waarde kan gespecificeerd worden al een variabele of constante met een bereik van 0-255.

```
analogWrite(pin, waarde); // schrijf 'waarde' naar analoge 'pin'
```

Een waarde van 0 geeft 0 V als output op de gespecificeerde pin. Een waarde van 255 geeft 5 V als output op de gespecificeerde pin. Voor waarden tussen 0 en 255 vindt er een evenredige pulsbreedte modulatie plaats, waarbij opgemerkt moet worden dat de breedte van de pulsen evenredig groter wordt naarmate de waarde stijgt.

Omdat dit een hardware functie is zal de uitgegeven pulsbreedte continue het zelfde zijn totdat er een nieuwe analogWrite wordt gedaan.
Het volgende voorbeeld leest een analoge waarde van een analoge INPUT pin. Vervolgens wordt deze waarde aangeboden aan een PWM OUTPUT pin. Let op de gelezen waarde is van 0 – 1023 maar de maximale aangeboden PWM waarde mag niet meer zijn dan 255. Daarom wordt de gelezen waarde gedeeld door 4:

```
int led = 10; // LED met 220 weerstand op pin 10
int pin = 0; // potentiometer op analoge pin 0
int value; // value om te lezen
void setup(){} // geen setup nodig
void loop()
{
value = analogRead(pin); // lees 'value' op 'pin'
value /= 4; // converteer 0-1023 to 0-255
analogWrite(led, value); // outputs PWM signaal naar led
}
```


delay(ms)

Wachtlus (pauze) weergegeven in milliseconden, waarbij de waarde 1000 gelijk staat aan 1 seconde.

```
delay(1000); // wacht een seconde
```

millis()

Laat zien hoeveel milliseconde het Arduino board in werking is na de start van het lopende programma. De weergave is een long variabele.

```
Looptijd = millis(); // looptijd wordt gevuld met millis()
```

Note: Het weer te geven getal zal na verloop van tijd een overflow veroorzaken. (Een reset naar nul), na ongeveer 9 uur.

min(x, y)

Bereken het minimum van twee getallen en geef het kleinste getal weer.

```
waarde = min(getal, 100); // 'waarde' is gelijk aan 100
// of kleiner dan 100 als ‘getal’
// kleiner dan 100 is
```

max(x, y)

Bereken het maximum van twee getallen en geef het grootste getal weer.

```
waarde = min(getal, 100); // 'waarde' is gelijk aan 100
// of hoger dan 100 als ‘getal’
// hoger dan 100 is
```


randomSeed(seed)

Maak een willekeurige waarde aan (random).

```
randomSeed(value);
```

De Arduino kan uit zichzelf geen random nummer creëren.
Daarvoor is een commando dat wel een “willekeurige” random waarde kan aanmaken. Let wel: Er is nooit sprake van een absolute willekeurige waarde. Het is een functie om te helpen om één of meerdere “willekeurige” waardes aan te maken. De syntax is:

RandomSeed()

random(max)

random(min, max)

De random functie maakt het ook mogelijk om waardes in een reeks aan te maken:

```
value = random(100, 200); // sets 'value' to a random
// number between 100-200
```
Opmerking: Laat bovenstaande code vooraf laten met de randomSeed() functie.
Het volgende voorbeeld maakt een willekeurige waarde aan tussen 0-255 en zet de aangemaakte waarde over naar een PWM pin:
```
int randNummer; // variabele om random waarde te bewaren
int led = 10; // LED met 220Ω weerstand op pin 10
void setup() {} // geen setup nodig
void loop()
{
randomSeed(millis()); // gebruik millis()
randNumber = random(255); // random nummer van 0-255
analogWrite(led, randNumber); // output PWM signaal
delay(500); // wacht halve seconde
}
```


Serial.begin(rate)

Open een seriële poort, zet de juiste baudrate om seriële data te kunnen verzenden. Een baudrate van 9600 wordt veel gebruikt maar andere snelheden zijn ook mogelijk.
```
void setup()
{
Serial.begin(9600); // open een seriele port
} // met een baudrate van 9600 bps
```
Opmerking: Wanneer gebruik wordt gemaakt van seriële communicatie op pin 0 (Rx) en pin1 (Tx) let er dan op dat deze niet tegelijkertijd gebruikt kunnen worden.

Serial.println(data)

Stuurt data naar de seriële poort tezamen met een carriage return en een line feed. Dit commando heeft dezelfde vorm als het commando Serial.print().
```
Serial.println(analogValue); // zend de waarde van een
// analoge waarde'analogValue'
```
Opmerking: Meer informatie over de verschillende van de Serial.println() en Serial.print() functies kijk op de Arduino website. www.arduino.cc en www.arduino.nu.

Het volgende voorbeeld leest de analoge waarde op pin 0 en zend deze data elke seconde naar de computer.
```
void setup()
{
Serial.begin(9600); // open poort naar 9600bps
}
void loop()
{
Serial.println(analogRead(0)); // zend analoge waarde
delay(1000); // wacht 1 seconde
}
```


Bijlage

Een simpel programma is het laten knipperen van een LED. Dat gaat bij de Arduino