5.0. Arduino Programmeren voor Beginners – Deel 5: Lussen

Dit is de vijfde les van Arduino Programmeren voor Beginners, waarin we gaan kijken hoe we met lussen kunnen gaan werken.
Dat klinkt natuurlijk raar, maar wat ik daarmee bedoel is dat we code gaan herhalen, bijvoorbeeld bij het aftellen of in afwachting van een conditie.

Dit zijn de zogenaamde “loops” zoals de “for”-loop, “while”-loop en de “do … while …”-loop. Deze loops hebben ook invloed op de eerdergenoemde “Control Flow” van ons programma bepalen – of te wel, bepalen in welke situatie, welk deel van het programma uitgevoerd gaat worden.

Inhoudsopgave

5.1. Het doel van Loops (lussen)

Voor we gaan werken met lussen (of: loops), is het misschien handig om een voorstelling te krijgen waar we deze loops zouden kunnen gebruiken.

De meest gebruikte “lus” of “loop” is wanneer een instructie vaak herhaald dient te worden, vaak op basis van tellen of aftellen.

Stel we hebben 5 lampen en elk heeft een eigen schakelaar. Om ze alle 5 aan te zetten moeten we dus:

Schakel Lamp 1 AAN,
Schakel Lamp 2 AAN,
Schakel Lamp 3 AAN,
Schakel Lamp 4 AAN,
Schakel Lamp 5 AAN.

We herhalen steeds dezelfde soort handeling, namelijk voor elke lamp de schakelaar op AAN zetten. We zouden dat ook als volgt kunnen beschrijven:

Tel van 1 tot 5, en voor elk nummer, zet de de Lamp AAN.

Tada! Onze eerste “loop” of lus – wat ons zelfs in dit simpele voorbeeld al werk bespaart, want we hoeven niet steeds een bijna dezelfde regel in te typen.

Bedenk maar eens als we dit voor 1.000 lampen zouden moeten doen!?
Niet alleen bespaard ons dit veel type werk, maar onze code wordt ook korter en veel beter leesbaar. Niet alleen dat: het aanpassen voor meer of minder lampen is een kwestie van een nummer veranderen en … de gecompileerde (vertaalde) code is vaak kleiner (neemt minder geheugen in beslag), en vaak zelfs sneller.

Loops worden gebruikt om bepaalde code meerdere keren te herhalen

Heel vaak gebruiken we in een loop een variabele zeker als we gaan tellen. Met dat in gedachten zou onze loop er zo uit kunnen zien:

```
Tel voor A = 1 to 5
  schakel Lamp A AAN
```

Dit soort “code” noemt men “Pseudo-code“.
Pseudocode is eigenlijk het opschrijven van instructies voor ons programma (b.v. op papier) zodat het beter leesbaar is in een meer menselijke taal, terwijl we toch constructies gebruiken die we in de programmeertaal gebruiken. Uiteraard kunnen we pseudo-code niet meteen in ons programma plakken maar het helpt ons om instructies uit te vogelen voor we aan de slag gaan met een echt programma.
Je mag deze naam weer vergeten als je wilt, het schrijven van pseudo-code zul je echter toch onbewust een keer gaan doen.

De “for” loop

In ons voorbeeld gebruikten we het “for” statement eigenlijk stiekem al. “for” is namelijk het Engelse woord voor “voor”.
Let er wel op dat tellen met een “for” loop is gebaseerd op tellen met gehele nummers!

Tel for A = 1 to 5
  schakel Lamp A AAN

Als we dit wat complexer opschrijven dan krijgen we:

for A = 1, en A moet kleiner of gelijk aan 5 zijn, en in elke ronde verhogen we A met 1
  schakel Lamp A AAN

Dit is eigenlijk precies zoals een for-loop in de C taal werkt, en dat ziet er zo uit:

```
for(A=1; A<=5; A++) {
  Serial.print("Schakel deze lamp AAN: ");
  Serial.println(A);
}
```

Laten we deze code eens gaan bekijken.

Het “for” statement is waarschijnlijk wel voor de hand liggend. De parameters die we nodig zijn voor het “for” statement zijn:

Start met “A = 1”

Vergeet niet dat “A <= 5”

Verhoog de waarde van A elke keer dat we door de lus gaan (A++)

Tellen in een “for” loop kan alleen gedaan worden met gehele nummers (byte, int. long, etc), of in andere woorden: het gebruikte data type voor de “teller” moet een enumeratie type zijn (tellen met hele nummers, met gelijke afstanden tussen nummers).

Omdat we dit zouden kunnen zien als parameters voor de “for” functie, moeten we er ronde haakjes omheen zetten (zoals we zagen bij b.v. Serial.print()).

Het volgende dat we zien is dat we het code blok die herhaald moet worden, tussen accolades gezet zijn – net zoals we dat b.v. ook bij het “if” statement zagen.

Alles in dit blok wordt herhaald voor elke stap van de “for” loop.

Natuurlijk moeten we hier weer die lastige “compound operators” tegen komen welke we in Deel 3 zagen.: A++”.
“A++” wil zeggen: “verhoog de waarde van A met 1 maar werk eerst nog met de oude waarde van A“.
Dus als we door een loop gaan, werken we dus eerst met de oude waarde van “A”, maar zodra we klaar zijn met de loop, zal A verhoogd zijn met 1.
Nu weet je misschien ook waarom ik geen fan ben van compound operators – erg verduidelijkend zijn ze niet he?

Gewoon zonder nadenken onthouden voor “for” loops:
In een “for” loop werk je altijd met

“++” (optellen) en
“—” (aftellen)

Ehm, zei je nu “aftellen” …? Ja zekers!
Een “for” loop kan ook aftellen, hierbij gebruiken we de “—” compound operator. Stel we draaien het eerdergenoemde voorbeeld om en we gaan aftellen van 1 tot 5 dan krijgen we dit:

```
for(A=5; A>=1; A--) {
  Serial.print("Schakel deze lamp AAN: ");
  Serial.println(A);
}
```

Zoals je ziet starten we nu met “A = 5”, en we willen dat A groter of gelijk aan 1 moet zijn. We tellen immers af in dit voorbeeld.

Nu hebben we nog 1 ding over het hoofd gezien … de definitie van de variabele A.

We kunnen “A” op de gebruikelijke manier definiëren, voor we de loop starten: int A;

```
int A;

for(A=1; A<=5; A++) {
  Serial.print("Schakel deze lamp AAN: ");
  Serial.println(A);
}
```

De alternatieve manier is om “A” in het “for” statement te definiëren, wat ideaal is als we “A” alleen maar in de loop gaan gebruiken (herinner je nog het onderwerp “scope”?). In code ziet er dat zo uit:

```
for(int A=1; A<=5; A++) {
  Serial.print("Schakel deze lamp AAN: ");
  Serial.println(A);
}
```

Zie je dat we “int A” in de parameter hebben geplaatst?
Dit is een relatief normale methode in de taal C, maar vanwege het “scope” principe zal deze variabele dus alleen in het code blok van het “for”-statement bekend zijn.

Als je het onderstaande voorbeeld met Arduino compiler probeert dan krijg je de melding “error: ‘A’ was not declared in this scope” (fout: ‘A’ was niet gedeclareerd in deze scope).

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    for(int A=1; A<=5; A++) {
6      Serial.print("Schakel deze lamp AAN: ");
7      Serial.println(A);
8    }
9
10   Serial.println(A); // deze regel gaat fout!
11 }
12
13 void loop() {
14   // leave empty for now
15 }
```

Zoals in Deel 3 vermeld dus, De “scope” van “A” geldt alleen maar voor de code blok van de “for”-loop. Dus “A” is onbekend op regel 10, en daarmee bedoelen we dat het niet gedefinieerd is (niet gedeclareerd).

De variabele voor de “teller” in onze “for”-loop kan in het “for” statement gedefinieerd worden – maar bedenk wel dat deze dus alleen in het code blok bekend is!

Als je vergeet de variabele voor de “for”-loop te definiëren, dan zal de Arduino dat automatisch voor je doen – het is echter een goede gewoonte om het correct te doen door de variabele vooraf, of in het “for” statement correct te definiëren!

Let dus goed op waar de variabele gebruikt moet worden en maak er een gewoonte van om de betreffende variabele correct te definiëren!

Laten we dit in een werkend Arduino voorbeeld zetten – ik heb optellen en aftellen erbij in begrepen:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    Serial.println("Optellen:");
6
7    for(int A=1; A<=5; A++) {
8      Serial.print("Schakel deze lamp AAN: ");
9      Serial.println(A);
10   }
```

```
11
12    Serial.println("Aftellen:");
13
14    for(int A=5; A>=1; A--) {
15      Serial.print("Schakel deze lamp AAN: ");
16      Serial.println(A);
17    }
18  }
19
20  void loop() {
21    // leave empty for now
22  }
```

De output ziet er dan zo uit:

Optellen:
Schakel deze lamp AAN: 1
Schakel deze lamp AAN: 2
Schakel deze lamp AAN: 3
Schakel deze lamp AAN: 4
Schakel deze lamp AAN: 5
Aftellen:
Schakel deze lamp AAN: 5
Schakel deze lamp AAN: 4
Schakel deze lamp AAN: 3
Schakel deze lamp AAN: 2
Schakel deze lamp AAN: 1

In de code blokken kunnen we dus de variabele die we in het “for” statement hebben gebruikt, gebruiken. Bedenk wel dat “A” in de eerste loop een andere “A” is dan die in de tweede loop. In de taal C, in tegenstelling tot een aantal andere programmeertalen, kunnen we A zelfs veranderen in de loop – maar dat kan ik niet aanraden, tenzij je wat meer ervaring het opgedaan en weet wat je doet.

Laten we dat eens als voorbeeld bekijken door de eerste loop te veranderen naar:

```
1  for(int A=1; A<=5; A++) {
2    Serial.print("Schakel deze lamp AAN: ");
3    Serial.println(A);
4    A = 7;
5  }
```

De output wordt dan:

Optellen:
Schakel deze lamp AAN: 1
Aftellen:
Schakel deze lamp AAN: 5
Schakel deze lamp AAN: 4
Schakel deze lamp AAN: 3
Schakel deze lamp AAN: 2
Schakel deze lamp AAN: 1

Dus in de eerste loop beginnen we met A=1, en de code in het code blok wordt uitgevoerd. Hierdoor wordt A echter ineens 7 wat groter is dan de gestelde limiet van 5 en dus ziet de “for” de loop als klaar en afgehandeld en verlaat de loop dus.
Vandaar dat we dus maar 1 regel van deze eerste loop zien verschijnen.

Elke keer dat we door een “loop” gaan, noemt met een iteratie,
dus ons voorbeeld had 2 “for” loops, elk met 5 “iteraties “.

Het nadeel van het veranderen van de tellende variabele is dat we hier mee in de problemen kunnen komen. Stel dat we de regel “A = 7;” verkeerd hadden ingetypt en daarvoor in de plaats “A = 1;” hadden geschreven?
De loop gaat nu oneindig door, want de eind conditie situatie wordt nooit bereikt.

Het veranderen van de tellende variabele van een loop in dezelfde loop is over het algemeen een slecht idee!

Mocht je dit willen gebruiken om een loop te verlaten, gebruik dan het “break” statement.
“break” verlaat de loop net zoals we dat bij “switch” zagen. Dit geldt overigens voor alle loops: “for”-, “while”- en “do … while …”-loops.

Met het “break” statement kun je een loop verlaten.

Bijvoorbeeld:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    for(int A=1; A<=5; A++) {
6      Serial.print("Schakel deze lamp AAN: ");
7      Serial.println(A);
8      if(A==3) {
9        break;
10     }
11   }
12 }
13
14 void loop() {
15   // leave empty for now
16 }
```

In onze voorbeelden hebben we het eenvoudig gehouden, we hebben met de nummers 1 en 5 gewerkt. Maar je hoeft niet te beginnen of te eindigen met het nummer 1. Het is zelfs zeer gebruikelijk om b.v. met het nummer “0” (nul) te beginnen. In principe kun je met elk geheel nummer beginnen en eindigen en er zij zelfs situaties waar een loop geen iteraties heeft, b.v. als je van nul tot nul gaat.

De “condities” in de “for”-loop hoeven overigens ook geen vaste nummers te zijn. Zoals we al weten kunnen variabelen de posities van een nummer innemen. Dus het volgende zou ook werken:

```
int Start = 1;
int Stop  = 5;

for(int A=Start; A<=Stop; A++) {
  Serial.print("Schakel deze lamp AAN: ");
  Serial.println(A);
}
```

Ik adviseer om nu eens een beetje te gaan prullen met de “for” loop – probeer maar eens wat waardes aan te passen, zodat je wat meer bekend wordt met deze loops.

“while” loops

Nu dat we de “for”-loop hebben bekeken waarbij de “range” voor het tellen meer voorgedefinieerd is, moeten we bedenken dat er ook loops zijn van een andere structuur waarbij we nog niet weten hoe ver we gaan tellen. Zoiets als dit:

Terwijl het buiten donker is, lichten AAN

Als eerste: “terwijl” vertaalt zich in het Engels naar “while”, dus onze regel wordt:

While het buiten donker is, lichten AAN

Omdat we niks tellen, kunnen we dit met en “for”-loop niet netjes oplossen – uiteraard zijn er trucs om dit in een “for”-loop op te lossen, maar dat moet dan met een wat slordige methode gedaan worden.

Dit is waar we de zogenaamde “while” loop gaan gebruiken, wat er ongeveer zo uit gaat zien:

```
While DonkerBuiten
  LichtenAan
```

Dus we kijken eerst of onze conditie, DonkerBuiten, waar is (true) en als dat het geval is, dan voeren we de code block (LichtenAan) uit. Na het uitvoeren van het code block gaan we weer naar “while” en testen we de conditie weer, enz. enz. tot dat de conditie niet meer waar is (false) en de loop (lus) verlaten wordt.

Een “while” loop controleert eerst of de conditie waar is,
en als dat het geval is, wordt de code block uitgevoerd, en start de loop opnieuw totdat de conditie false oplevert en de loop verlaten wordt …

Je zou dit kunnen zien als een zich eindeloos herhalende “if”. Het blijft zicht herhalen, zolang de conditie true oplevert. Dit in tegenstelling tot het “if” statement, welke maar èèn keer doorlopen wordt.

Dit zou er dus ongeveer zo uit kunnen zien:

```
while (DonkerBuiten==true) {
  Serial.println("Lampen AAN");
}
```

Zolang de stelling “DonkerBuiten” == true een “true” oplevert, zal de Arduino “Lampen AAN” blijven weergeven.

Echter … we kunnen dit voorbeeld ook eenvoudiger schrijven omdat “DonkerBuiten” zelf al een boolean is en true of false is.

```
while (DonkerBuiten) {
  Serial.println("Lampen AAN");
}
```

Natuurlijk werkt dat niet als je met nummers en tekst gaat werken, omdat deze natuurlijk meerdere verschillende waarden zouden kunnen zijn.

Er is echter iets waar we even naar moeten kijken bij een “while”-loop.
Als de waarde “DonkerBuiten” namelijk nooit veranderd, dan blijft de loop oneindig doorgaan …

Er zijn twee mogelijkheden om “DonkerBuiten” te veranderen …

De eerste manier (intern) is wanneer de waarde binnen het code block veranderd wordt.
Bijvoorbeeld door een berekening of door te tellen (voor dat laatste is een “for”-loop natuurlijk vaak handiger):

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    // define our variable
```

```
6   int A = 1;
7
8   while(A<=5) {
9     Serial.print("Lamp AAN: ");
10    Serial.println(A);
11    A = A + 1;
12  }
13  }
14
15  void loop() {
16   // leave empty for now
17  }
```

Dit voorbeeld doet hetzelfde als een “for”-loop zoals hieronder, wat er natuurlijk veel efficiënter uit ziet:

```
1   void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    for(int A=1; A<=5; A++) {
6      Serial.print("Lamp AAN: ");
7      Serial.println(A);
8    }
9   }
10
11  void loop() {
12   // leave empty for now
13  }
```

De andere manier is als de waarde van buitenaf (extern) veranderd wordt, bijvoorbeeld door het lezen van een licht-sensor of een schakelaar.

Nu hebben we nog niets met schakelaars en sensoren gedaan, en dat laat ik ook even zo voor onze volgende serie.

Vergeet niet dat als de conditie van de”while” loop altijd false oplevert,
dat de loop nooit uitgevoerd gaat worden!

“do … while …” loops

De “do … while …” (Engels voor: doe … terwijl …) loop is eigenlijk hetzelfde als de zonet besproken “while” loop, alleen draaien we het testen van de conditie om. Eerst voeren we het code blok uit en daarna pas testen we de conditie om te bepalen dat de loop nogmaals doorlopen dient te worden.

Een “do … while …” voert de instructies in het code blok eerst uit
en kijkt daarna pas of de conditie waar is. Indien de conditie waar is, wordt de loop herhaald, indien de conditie niet waar is, dan wordt de loop verlaten.

De notatie lijkt op de “while” loop, maar dan achterstevoren:

```
do {
  // doe iets
} while (conditie);
```

We zien dat we met het “do” statement starten, gevolgd door een code blok en zien als laatste pas het “while” statement.

Een toepassing kan bijvoorbeeld zijn als er eerst een berekening gedaan moet worden, welke mogelijk herhaald dient te worden tot de conditie niet meer een true oplevert.

Hier een voorbeeld van de “while”-loop en de “do … while …”-loop welke beide hetzelfde doen:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    int A = 1;
6
7    while(A<=5) {
8      Serial.println("WHILE: A is nog steeds <= 5");
9      A = A + 1;
10   }
11
12   A = 1;
13
14   do {
15     Serial.println("DO WHILE: A is nog steeds <= 5");
16     A = A + 1;
```

```
17    } while (A<=5);
18    }
19
20    void loop() {
21      // leave empty for now
22    }
```

De output ziet er zo uit:

```
WHILE: A is nog steeds <= 5
WHILE: A is nog steeds <= 5
WHILE: A is nog steeds <= 5
WHILE: A is nog steeds <= 5
WHILE: A is nog steeds <= 5
DO WHILE: A is nog steeds <= 5
DO WHILE: A is nog steeds <= 5
DO WHILE: A is nog steeds <= 5
DO WHILE: A is nog steeds <= 5
DO WHILE: A is nog steeds <= 5
```

De “do … while …”-loop kan natuurlijk ook lekker fout gaan, met name omdat we de conditie testen na het uitvoeren van de instructies van het code blok. Kun je bedenken wat er gebeurt als we regels 5 en 12 veranderen naar: A = 10; ?

De “while”-loop wordt niet uitgevoerd en in dit geval is dat goed.
Echter de “do … while …”-loop wordt wel èèn keer uitgevoerd, maar dat is helaas fout: “DO WHILE: A is nog steeds <= 5”.

Beiden soorten loops hebben hun toepassing.
Controleer goed welk resultaat je verwacht!

Controleer conditie eerst en doe de code blok daarna? (while-loop)

Doe de code blok en controleer de conditie daarna? (do-while-loop)

We moeten dus goed opletten of we een “while” of een “do … while …” nodig hebben voor onze situatie. Ik denk dat de “while”-loop het meeste gebruikt wordt.

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 6: Functies