4.1. Deel 4: Arduino Programmeren voor Beginners – Beslissingen

Dit is de vierde les van Arduino Programmeren voor Beginners.
In deze les gaan we kijken naar het maken van beslissingen in ons programma.

Dit zijn de zogenaamde “if…then..,” (als…dan…) situaties, en omdat we in ons programma informatie vergelijken om vervolgens beslissingen te maken, gaat dit een belangrijk onderdeel worden. Deze beslissingen gaan de zogenaamde “Control Flow” van ons programma bepalen – of te wel, bepalen in welke situatie, welk deel van het programma uitgevoerd gaat worden.

Inhoudsopgave

4.2. Beslissingen maken in ons Arduino Programma

In het voorgaande hoofdstuk hebben we gekeken naar de zogenaamde Comparison Operators (vergelijking operators), waarmee we informatie (data) kunnen vergelijken. We weten dat een dergelijk vergelijking een booleaans antwoord gaat geven, dus “true” (waar) of “false” (niet waar). Iets wat we in dit deel gaan gebruiken om beslissingen (decisions) te maken in ons programma.

We hebben ook naar de zogenaamde Booleaanse Operators gekeken en deze zijn een belangrijke aanvulling in deze les.

Omdat onze programma’s vaak kijken naar verschillende situaties, is het van belang dat we vergelijkingen kunnen maken en aan de hand daarvan beslissingen kunnen maken. Dat uit zich in het bepalen wanneer, welk deel van onze code uitgevoerd gaat worden. Dit heet in het Engels “control flow” of in andere worden: controleren hoe we door een programma gaan.

Als simpel voorbeeld zou ons programma lichten aan of uit kunnen zetten, afhankelijk van het feit of het nu buiten licht of donker is.
In ons programma moeten we dus eerst kijken of het buiten donker is, is dat het geval dan moeten we de code uitvoeren die de lampen aan gaan zetten. Als het echter niet donker is, dan moeten we de code uitvoeren die de lampen uit zetten.

Dit zou je kunnen lezen als: als … dan …
Dus als iets waar is, doen dan dit, als iets niet waar is doe dan dat.
Het “als … dan …” Vertaald zich in het Engels naar “if … then …”, een constructie die je in heel veel programmeertalen terug gaat vinden.

If … then … of te wel “Als … dan …”

Dit is echt een van de meest gebruikte manieren om beslissingen in jouw programma te maken.
Een stukje van jouw programma wordt alleen maar uitgevoerd als een bepaalde stelling waar is – dit heet conditionele instructies of in the Engels een “Conditional Statements“. Dus instructies die alleen worden uitgevoerd onder bepaalde condities.

Laten we eens gaan kijken naar 2 eenvoudige voorbeelden.

Stel we hebben lampen die automatisch aan gaan als het buiten donker wordt.
De conditionele instructie wordt dan:

als BuitenDonker dan ZetLichtenAan

We zagen al dat “als..dan..” in het Engels “id…then…” is, en om daar aan te wennen gaan we de Engels variant gebruiken in onze tekst:

if GeldInPortemonnee=0 then GeldGaanPinnen

Dus als onze Portemonnee leeg is (GeldInPortemonnee=0) dan moeten we even gaan pinnen om weer geld in onze Portemonnee te krijgen.

Ik denk dat je nu wel situaties kunt bedenken waarbij een “if … then …” in jouw programma handig kan of zelfs nodig kan zijn. Je wilt uiteindelijk dat jouw programma correct reageert op een mogelijk veranderende situatie, of klaar zijn voor gebruik in meerdere situatie. Het deel dat anders kan zijn of kan veranderen kan een waarde of variabele zijn (van bijvoorbeeld een licht sensor of knop druk).

De juiste notatie (“notatie” in het Engels noemen we “syntax”) is:

```
1  if ( conditie ) {
2    // wat gedaan moet worden als de conditie waar (true) is
3  }
4  else {
5    // wat gedaan moet worden als de conditie niet waar (false) is
6  }
```

We zien weer de zogenaamde code blokken die we eerder hebben gezien – tussen de accolades. Elk blok hoort bij een conditie.

Het “else” statement wordt altijd aan het einde van een “if … then …” geplaatst, dus als de laatste “conditie” wat wordt uitgevoerd als alle condities falen …

Wat verder opvalt is het “else” woord in regel 4. Het woord “else” is het Engelse woord voor “anders”. Dus “als iets waar is dan doe dit anders doe dat”. Het blok dat volgt na “else” is het stukje code wat uitgevoerd als alle andere de conditie(s) falen.
Dit deel (regels 4, 5, en 6) is echter geheel optioneel en hoeft dus niet gebruikt te worden, wat de notatie kan vereenvoudigen naar:

```
1  if ( conditie ) {
2    // wat gedaan moet worden als de conditie waar (true) is
3  }
```

De “conditie” is over het algemeen het resultaat van een comparison (vergelijking) operator wat, zoals we al weten, altijd een true of false oplevert. Kijk eventueel terug naar het voorgaande hoofdstuk. Voor het gemak heb ik de tabel hier nog eens geplaatst;

Comparison Operatos	
Symbool	Doel
==	is gelijk aan
!=	is niet gelijk aan
<	is kleiner dan
>	is groter dan
<=	is kleiner dan of gelijk aan
>=	is groter dan of gelijk aan

De meest voorkomende fout bij Comparison Operators is dat men
het symbool “=” gebruikt (toewijzing!) in plaats van het “==” teken als we kijken of twee waarden gelijk zijn.

Overigens, nog een tip die in de toekomst handig gaat zijn: een “if…then…” beslist op basis van een conditie die true of false kan zijn – dus een boolean! Dit wil dus ook zeggen dat we een boolean waarde kunnen gebruiken in plaats van een conditie waarin we een vergelijk zetten. Onderstaande voorbeeld zal bijvoorbeeld altijd de “if”-blok uitvoeren:

```
1  if ( true ) {
2    // doen als de conditie waar is
3  }
```

OK, laten we eens door een werkend programma gaan, en in dit geval borduren we verder op ons geldprobleem.

We gaan kijken hoeveel geld we op zak hebben en hoeveel geld we gespaard hebben. Als we minder dan € 5 hebben dan willen we een waarschuwing zien zodat we weten dat we niet al te veel geld meer hebben.

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    // define our variables
6    int ZakGeld;
7    int SpaarGeld;
8    int AlMijnGeld;
9
10   // assign the values
11   ZakGeld = 4;
12   SpaarGeld = 12;
13   AlMijnGeld = ZakGeld + SpaarGeld;
14
15   // print the values to the serial monitor
16   Serial.print("Geld op zak = ");
17   Serial.println(ZakGeld);
18
19   Serial.print("Spaargeld = ");
20   Serial.println(SpaarGeld);
21
22   Serial.print("Al mijn Geld = ");
23   Serial.println(AlMijnGeld);
24
25   if(AlMijnGeld<5) {
26     Serial.println("Oh oh,... we hebben maar weinig geld!!");
27   }
```

```
28    else {
29      Serial.println("Geen probleem, we hebben genoeg geld.");
30    }
31  }
32
33  void loop() {
34    // even leeg laten
35  }
```

Een deel van deze code hebben we al eerder gezien. Het beslissing-stuk vinden we vanaf regel 25 waar we gaan kijken of we nog genoeg geld hebben … dus we gaan kijken of de variabele “AlMijnGeld” minder dan (<) 5 euro is. Als we minder dan 5 Euro hebben, dan willen we een waarschuwing zien “Oh oh,… we hebben maar weinig geld!!“. Mochten we meer dan 5 Euro hebben, dan is alles OK en willen we de melding “Geen probleem, we hebben genoeg geld.” zien.

Kopieer de code en plak deze in de Arduino IDE, compileer het en stuur het naar de Arduino.

Omdat de conditie “false” is (4+12=16 en 16 is groter dan 5) zal het “if” code blokje overgeslagen worden en gaat de Arduino meteen verder met het “else” stuk:

Geld op zak = 4
Spaargeld = 12
Al mijn Geld = 16
Geen probleem, we hebben genoeg geld.

Laten we nu regel 12 veranderen en ons spaargeld op zetten: SpaarGeld = 0;.
Compileer en upload het programma opnieuw naar de Arduio.

De conditie in regel 25 is nu WEL WAAR, dus de “if” blok wordt uitgevoerd en het “else” blok wordt nu overgeslagen wat een output geeft als:

Geld op zak = 4
Spaargeld = 0
Al mijn Geld = 4
Oh oh,... we hebben maar weinig geld!!

We kunnen ook een voorbeeld bedenken waarbij we het “else” stuk niet gebruiken. Stel je wilt alleen een waarschuwing zien als het foute boel is met ons geld, maar als alles OK is, dan willen we niks zien. Zo’n voorbeeld zou er zo uit kunnen zien:

```
1  void setup() {
2   // set the speed for the serial monitor:
3   Serial.begin(9600);
4
5   // define our variables
6   int ZakGeld;
7   int SpaarGeld;
8   int AlMijnGeld;
9
10  // assign the values
11  ZakGeld = 4;
12  SpaarGeld = 12;
13  AlMijnGeld = ZakGeld + SpaarGeld;
14
15  // print the values to the serial monitor
16  Serial.print("Geld op zak = ");
17  Serial.println(ZakGeld);
18
19  Serial.print("Spaargeld = ");
20  Serial.println(SpaarGeld);
21
22  Serial.print("Al mijn Geld = ");
23  Serial.println(AlMijnGeld);
24
25  if(AlMijnGeld<5) {
26    Serial.println("Oh oh,... we hebben maar weinig geld!!");
27  }
28  }
29
30  void loop() {
31   // even leeg laten
32  }
```

Het “if … then …” statement heeft echter ook nog eens de mogelijkheid om meerdere condities samen te voegen.

Als voorbeeld:

if AlMijnGeld=0 then paniek anders als AlMijnGeld<5 dan haal meer geld ander niks aan het handje.We stapelen zeg maar meerdere “if .. then … else if … then … else if … then” op, en jouw Arduino zal netjes door alle condities lopen tot het een conditie vindt die waar is en voert dan be bijbehorende code uit en verlaat dan de hele “if .. then ..” constructie!

“if … then …” statements kunnen opgebouwd worden uit meerdere “if … then …” constructies.
Als in zo’n constructie een conditie gevonden wordt die WAAR is, dan wordt bijbehorende code blok uitgevoerd en de gehele constructie verlaten. De andere “if..then..” of “else” worden dan dus niet meer bekeken!

Een voorbeeld:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    // define our variables
6    int ZakGeld;
7    int SpaarGeld;
8    int AlMijnGeld;
9
10   // assign the values
11   ZakGeld = 4;
12   SpaarGeld = 12;
13   AlMijnGeld = ZakGeld + SpaarGeld;
14
15   // print the values to the serial monitor
16   Serial.print("Geld op zak = ");
17   Serial.println(ZakGeld);
18
19   Serial.print("Spaargeld = ");
20   Serial.println(SpaarGeld);
21
22   Serial.print("Al mijn Geld = ");
23   Serial.println(AlMijnGeld);
24
25   if(AlMijnGeld==0) {
26     Serial.println("PANIEK!!");
27   } else if(AlMijnGeld<5) {
28     Serial.println("Oh oh,... we hebben maar weinig geld!!");
29   } else if(AlMijnGeld>10) {
30     Serial.println("Woohoo - we zijn rijk!!");
31   } else {
32     Serial.println("Geen probleem, we hebben genoeg geld.");
33   }
34 }
35
36 void loop() {
37   // leave empty for now
38 }
```

Deze code zal:
– “PANIEK!!” weergeven als AlMijnGeld = 0, OF
– “Oh oh,… we hebben maar weinig geld!!” weergeven als AlMijnGeld kleiner is dan 5, OF
– “Woohoo – we zijn rijk!!” weergeven als AlMijnGeld groter is dan 10, OF
– in alle andere situaties “Geen probleem, we hebben genoeg geld.” weergeven.

Meerdere “if … then … else if … then”

Als we naar het voorgaande voorbeeld kijken, dan moet je goed opletten wat hier gebeurt!

Een “if … then ..” statement wordt verlaten bij het vinden van een “true” conditie.

Alle volgende “else if ….” en “else” statements worden genegeerd !!!!

In on voorbeeld is AlMijnGeld = 16, dus de eerste 2 condities FALEN.
De derde conditie is echter WAAR, dus condition SLAAGT, en dus zal de melding “Woohoo – we zijn rijk!!” zichtbaar worden gemaakt.
Omdat deze conditie WAAR is, zal het “else” blok genegeerd worden. Logisch?

OK, laten we regels 11 en 12 aanpassen:

```
11  ZakGeld = 0;
12  SpaarGeld = 0;
```

Dus AlMijnGeld zal dus nul (0) gaan worden in ons programma.

Maar als we nu naar onze “if … then …” statements gaan kijken dan zien we iets interessants wat een fout (bug) in ons programma kan veroorzaken.

Kijk maar … dus AlMijnGeld = 0.
Dus de eerste conditie SLAAGT, en het bericht “PANIKE!!” verschijnt.
Echter,… conditie 2 (AlMijnGeld kleiner dan 5) is ook WAAR, maar zal nooit worden uitgevoerd.

Als namelijk een conditie wordt gevonden die WAAR is (true), dan wordt de rest van de “if … then …” gezien als “klaar” en zullen de andere condities niet eens getest worden en dat is express zo gedaan.

Als je dit niet zou weten, dan zou je ook de melding “Oh oh,… we hebben maar weinig geld!!” verwachten.

Dit komt omdat we condities hebben gemaakt die elkaar overlappen. Daarmee bedoelen we dat 2 of meer condities WAAR kunnen zijn onder bepaalde omstandigheden – en dat kan soms onwenselijk zijn.

Maar wat moeten we dan doen om beide meldingen toch zichtbaar te krijgen?
In zo’n geval moeten we de “if…then…” uitelkaar halen en afzonderlijk opzetten, maar dat kan weer problemen veroorzaken als je een “else” aan het einde hebt.

Hier een voorbeeld hoe dat er uit zou kunnen zien:

```
25  if(AlMijnGeld==0) {
26    Serial.println("PANIEK!!");
27   }
28
29   if(AlMijnGeld<5) {
30    Serial.println("Oh oh,... we hebben maar weinig geld!!");
31   }
32
33   if(AlMijnGeld>10) {
34    Serial.println("Woohoo - we zijn rijk!!");
35   }
```

Dit werkt, maar … omdat we in onze oorspronkelijk code een “else” hebben voor de overige scenario’s, lopen we tegen een probleem aan. Waar zouden we deze “else” dan moeten zetten?

In ons oorspronkelijke programma zeiden we:

If AlMijnGeld == 0 then “Paniek”, else if AlMijnGeld < 5 then “Oh Oh”, else if AlMijnGeld > 10 then “Woohoo”, else “Geen probleem”.

Maar door het opbreken van de “if … then …” constructie, hebben we alles kunnen vatten, behalve wat er na het laatste “else” statement gebeurt:

If AlMijnGeld == 0 then “Paniek”.
If AlMijnGeld < 5 then “Oh Oh”.
If AlMijnGeld > 10 then “Woohoo”.

We kunnen geen “else” toevoegen, bij welke van de 3 dan ook, anders zou deze namelijk uitgevoerd kunnen worden op ongepaste momenten. Kijk hier maar eens naar:

If AlMijnGeld == 0 then “Paniek” else “Geen probleem”.
If AlMijnGeld < 5 then “Oh Oh” else “Geen probleem”..
If AlMijnGeld > 10 then “Woohoo” else “Geen probleem”.

Als AlMijnGeld nou 8 zou zijn, dan zouden we drie keer de melding “Geen probleem” zien omdat alle 3 de condites falen.
Als AlMijnGeld 0 (nul) zou zijn, dan zien we “Paniek” en twee keer “Geen Probleem” – en dat is ook niet correct.
Als AlMijnGeld 4 zou zijn, Dan zien we “Geen probleem” twee keer, en èèn keer “Oh Oh” – en dat klopt ook niet he?

Dus waar zit nou het gat waarbij de oorspronkelijke “else” wel uitgevoerd zou worden?

Als we naar onze condities gaan kijken dan zie we een bereik voor iedere conditie, waarbij een aantal daarvan elkaar zelf overlappen.
De waarde 0 (nul) en waarden kleiner dan 5 worden door de eerste 2 condities afgehandeld – en ze

overlappen elkaar, nul is immers ook kleiner dan 5.
Waarden groter dan 10 worden in de derde conditie gevangen.

Dus waar zit nu het gat waar “else” de zaak afvangt?

Complexere “if…then…” condities

De waarden groter of gelijk aan (>=) 5 EN kleiner of gelijk aan (<=) 10 worden niet afgehandeld – dit is ons “else” gat.

Stap voor Stap, even uitgaande van gehele nummers, dus zonder cijfers achter de komma:

Conditie 1 zorgt voor “0” (nul).
Conditie 2 zorgt voor “0, 1, 2, 3, 4” (kleiner dan 5 – dus dat wil zeggen dat 5 niet mee telt!).
Conditie 3 zorgt voor “11, 12, 13, … oneindig” (groter dan 10 – dus 10 telt hier niet mee!).

Zie je het gat nu? Het gat is “5 ,6 ,7 ,8 ,9 ,10”.

Dus alles groter of gelijk aan (>=) 5 EN kleiner of gelijk aan (<=) 10.

Dit is het punt waar onze Boolean Operators van pas komen:

Boolean Operator	
Symbool	Doel
&&	AND (en)
||	OR (of)
!	NOT (niet)

Het gaat bestaat eigenlijk uit twee condities.
De eerste is conditie is dat het getal groter of gelijk aan 5 moet zijn, dus: AlMijnGeld >= 5 .
De tweede conditie is dat het getal kleiner of gelijk aan “10” moet zijn, dus: AlMijnGeld <= 10 .

Maar hoe plakken we beiden condities nou bij elkaar als een enkele conditie? Beiden condities moeten immers gelden.

De eerste optie is een “if…then…” in een “if…then”, bij voorbeeld:

```
if(AlmijnGeld>=5) {
  if(AlMijnGeld<=10) {
    // doe dit
  }
}
```

Uiteraard werkt dit, maar dit wordt erg snel erg rommelig.

Laten we even naar de Boolean Operator tabel kijken, en lees dan deze tekst eens: het gat geld voor getallen >=5 en (and) <=10.

Het lampje gaat branden? De oplossing van ons probleem ligt in het woord “en” (and).

Kijk maar eens naar deze tabel van ons vorige hoofdstuk:

And Resultaten		
Conditie1	Conditie2	Resultaat van Conditie1 EN (and) Conditie2
true	true	= true
true	false	= false
false	true	= false
false	false	= false

Dus als beide condities waar zijn, hebben we ons gat, en in code met een boolean operator ziet dat er zo uit: AlMijnGeld >= 5 && AlMijnGeld <= 10.

Als je het zo schrijft kan dit wat problemen opleveren omdat we niet precies weten in welke volgorde dit wordt uitgevoerd. Eerst de “>=” en “<=” of eerst de “&&” (and)? De Arduino kan het wel goed afwerken hoor, maar je begrijpt dat leesbaarheid niet geweldig is. Overigens: andere programmeertalen kunnen hier ook problemen veroorzaken of dit zelfs niet een accepteren.

Om het leesbaar te houden, maar ook om er zeker van de te zijn dat alles juist wordt uitgevoerd, gaan we haakjes gebruiken.
Code die we tussen ronde haakjes zetten wordt namelijk als eerste uitgevoerd en daarna pas de rest, in dit geval dus het “&&” (and) teken.

Code tussen ronde haakjes wordt als eerste uitgevoerd ...

De nette notatie is dus: (AlMijnGeld >= 5) && (AlMijnGeld <= 10)

Als het programma draait dan wordt eerst gekeken wat “AlMijnGeld >= 5” en “AlMijnGeld <= 10” oplevert. De uitkomst van ieder vervangt de individuele conditie tussen haakjes en daarna wordt de rest van de regel pas gedaan.

Ter illustratie, stel AlMijnGeld = 6:

Stap 1: (AlMijnGeld >= 5) && (AlMijnGeld <= 10) en we weten dat “AlMijnGeld=6”
Stap 2: (true) && (AlMijnGeld <= 10)
Stap 3: (true) && (true)

Wat resulteerd in true && true wat dus “true” oplevert (zie tabel).

Onze aangepaste code wordt dus:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    // define our variables
6    int ZakGeld;
7    int SpaarGeld;
8    int AlMijnGeld;
9
10   // assign the values
11   ZakGeld = 0;
12   SpaarGeld = 4;
13   AlMijnGeld = ZakGeld + SpaarGeld;
14
15   // print the values to the serial monitor
16   Serial.print("Geld op zak = ");
17   Serial.println(ZakGeld);
18
19   Serial.print("Spaargeld = ");
20   Serial.println(SpaarGeld);
21
22   Serial.print("Al mij geld = ");
23   Serial.println(AlMijnGeld);
24
25   if(AlMijnGeld==0) {
26     Serial.println("PANIEK!!");
27   }
28
29   if(AlMijnGeld<5) {
30     Serial.println("Oh oh,... we hebben maar weinig geld!!");
31   }
32
33   if(AlMijnGeld>10) {
34     Serial.println("Woohoo - we zijn rijk!!");
35   }
36
37   if( (AlMijnGeld>=5) && (AlMijnGeld<=10) ){
38     Serial.println("Geen probleem, we hebben genoeg geld.");
39   }
40 }
41
42 void loop() {
43   // leave empty for now
44 }
```

Ik raad zeker aan om hier mee te gaan spelen. Rommel wat met de waarden in regels 11 en 12, en pas de condities vanaf regel 25 eens een beetje aan.

4.3. Het gebruik van ronde haakjes

We gaan toch nog eens wat beter kijken naar het gebruik van haakjes, omdat ze een belangrijke rol spelen in complexe berekeningen en vergelijking, en dan met name de haakjes tussen haakje. Dit noemt men “nested brackets”.

Als condities erg complex worden, dan kan het gebeuren dat we haakjes veel moeten gebruiken om ervoor te zorgen dat het e.e.a. in de juiste volgorde gedaan gaat worden door de computer. Uiteraard geldt dit ook voor berekeningen.

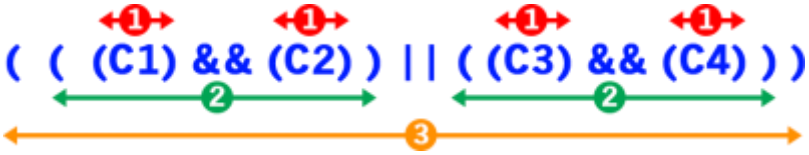
Laten we eens naar een voorbeeld gaan kijken: (((c1) && (c2)) || ((c3) && (c4)))
Elk c-nummer representeert een conditie.

De computer (Arduino) zal eerst de condities of berekeningen uitvoeren die het “diepst” zitten – waar dus de grootste hoeveelheid haakjes omheen staan. In onderstaande tekening zijn dat de condities met 1 gemarkeerd. Dus de condities c1, c2, c3 en c4. Deze worden dan vervangen door de uitkomst van deze condites waardoor het zoiets wordt als dit: ((u1 && u2) || (u3 && u4))
(waarbij het u-nummer staat voor de uitkomst van een conditie)

Vervolgens gaat de computer een niveau naar buiten, dus weer (simpel gezegd) de meeste haakjes omheen staan en welke in de tekening met 2 is gemarkeerd: (u12 || u34).
(waar u12 de uitkomst is van u1 && u2, en u34 de uitkomst van u3 && u4).

En uiteindelijk zal de computer aankomen bij de laatste haakjes (3 in de tekening), wat resulteert in: u1234
(waarbij u1234 de uitkomst is van u12 || u34).

Visueel weergegeven zou dat er zo uit kunnen zien:



Haakjes gebruiken – Verwerkt van binnen naar buiten

Nu snap ik helemaal dat dit misschien iets over je hoofd gaat, laten we dus naar een rekenvoorbeeld gaan kijken.

N.b.: Het voorbeeld hieronder is slechts ter illustratie, de Arduino zou dit zonder haakjes ook goed doen.

A = 3 * 4 + 12 / 4 + 6 * 3

Als we geen prioriteiten geven aan de reken tekens, dan zou het antwoord hiervan kunnen zijn:

A = 12 + 12 / 4 + 6 * 3
A = 24 / 4 + 6 * 3
A = 6 + 6 * 3
A = 12 * 3
A = 36

Maar we kunnen het ook anders doen waardoor we een ander antwoord krijgen:

A = 12 + 12 / 4 + 6 * 3
A = 12 + 12 / 10 * 3
A = 12 + 12 / 30
A = 24 / 30
A = 0.8

Je raadt het al: beiden zijn fout. Bij normale berekeningen weet de computer natuurlijk wel in welke volgorde het e.e.a. gedaan moet worden. Laten we dat eens illustreren met haakjes.

Let op: bij rekenen gaat vermenigvuldigen en delen voor optellen en aftrekken!

A = ((3 * 4) + (12 / 4) + (6 * 3))

Als we nu de volgorde van haakjes afhandelen gebruiken, waarbij de binnenste of diepste haakjes eerst gedaan worden, dan krijgen we:

A = ((12) + (12 / 4) + (6 * 3))
A = ((12) + (3) + (6 * 3))
A = ((12) + (3) + (18))
A = (15 + 18)
A = 33

En dit is het juiste antwoord – je ziet dus dat de volgorde van groot belang kan zijn.
Het gebruik van haakjes forceert dit zelfs en kan leesbaarheid bevorderen.

Code tussen haakjes, wordt eerst geëvalueerd. Van de diepste haakjes tot de uiterste haakjes, to er geen haakjes meer over zijn.

Switch … case …

Het “switch … case …” statement, is een soort super “if…then…” variant die niet vaak gebruikt wordt maar wel erg veel werk kan besparen.

Het vervangt een groep “if…then…” statements in een bepaald aantal situaties en heeft dus ook invloed op de control the flow van programma’s, waarbij bepaalde code wordt uitgevoerd onder bepaalde condities.

Het “if” statement staat toe dat we condities kunnen gebruiken m.b.v. vergelijking en booleans.
Het “switch” statement echter, werkt alleen maar met “equal to” of te wel “gelijk aan” situaties.

Laten we eens kijken naar een “if” voorbeeld, welke we later kunnen vervangen met een switch statement.

1	void setup() {
2	// set the speed for the serial monitor:
3	Serial.begin(9600);
4	
5	// define our variable
6	int A = 5;
7	

```
8  if(A==1) {
9    Serial.println("A = 1");
10 }
11 else if(A==2) {
12   Serial.println("A = 2");
13 }
14 else if(A==3) {
15   Serial.println("A = 3");
16 }
17 else if(A==4) {
18   Serial.println("A = 4");
19 }
20 else if(A==5) {
21   Serial.println("A = 5");
22 }
23 else if(A==6) {
24   Serial.println("A = 6");
25 }
26 else if(A==7) {
27   Serial.println("A = 7");
28 }
29 else if(A==8) {
30   Serial.println("A = 8");
31 }
32 else if(A==9) {
33   Serial.println("A = 9");
34 }
35 else if(A==10) {
36   Serial.println("A = 10");
37 }
38 else {
39   Serial.println("A is kleiner dan 1 of groter dan 10");
40 }
41 }
42
43 void loop() {
44  // leave empty for now
45 }
```

Deze lange code geeft het antwoord: A = 5

Maar dit is wel een hoop “if” statements, nietwaar?
En ja, dit kan efficiënter worden geschreven, en wel met het “switch” statement.

Switch

Het formaat hoe we “switch” gebruiken ziet er zo uit:

```
switch (variabele) {
  case waarde1:
   // doe dit als variable gelijk is aan waarde1
   break;
  case waarde2:
   // doe dit als variable gelijk is aan waarde2
   break;

  ... // etc.

  default:
   // optioneel: doe dit als geen van de stellingen waar is
  break;
 }
```

Dus het “switch” statement, neemt de huidige waarde van de “variabele” en vergelijkt het met de waarde die bij iedere “case” statement staat, om te bepalen of de daaropvolgende code moet uitvoeren. De optionele waarde “default:” (Engels voor “standaard”) geeft aan welke code wordt uitgevoerd als geen van de stellingen waar zijn. – dit hoeft dus niet gebruikt te worden in een switch statement.

Laten we de voorgaande code eens met een “switch” statement doen:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    // define our variable
6    int A = 5;
7
8    switch(A) {
9      case 1: Serial.println("A = 1");
10     case 2: Serial.println("A = 2");
11     case 3: Serial.println("A = 3");
12     case 4: Serial.println("A = 4");
13     case 5: Serial.println("A = 5");
14     case 6: Serial.println("A = 6");
15     case 7: Serial.println("A = 7");
16     case 8: Serial.println("A = 8");
```

```
17    case 9: Serial.println("A = 9");
18    case 10: Serial.println("A = 10");
19    default: Serial.println("A is kleiner dan 1 of groter dan 10");
20   }
21  }
22
23  void loop() {
24   // leave empty for now
25  }
```

In deze code heb ik expres een fout gemaakt, om een klein verschil met “if” aan tegeven!

De output is namelijk onverwacht:

A = 5
A = 6
A = 7
A = 8
A = 9
A = 10
A is kleiner dan 1 of groter dan 10

De reden dat dit fout gaat is omdat het “switch” statement zal zoeken naar een “case” die true is en zal de opvolgende code gaan uitvoeren.
omdat A=5, worden alle statements vanaf “case 5:” uitgevoerd.

Dit kan zeer handig zijn in bepaalde situaties, maar in de meeste gevallen willen we dat de “switch” statement stopt nadat het “case 5:” heeft gevonden. Om dat te bewerkstelligen, hebben we het “break” statement nodig aan het einde van iedere “case”.

Vergeet het “break” statement niet aan het einde van elke “case” als je de code uitvoering wil beperken tot alleen deze case.

Onze verbeterde code ziet er daarom dan ook zo uit:

```
1   void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    // define our variable
6    int A = 5;
7
8    switch(A) {
9      case 1: Serial.println("A = 1");
10           break;
11     case 2: Serial.println("A = 2");
```

```
12          break;
13      case 3: Serial.println("A = 3");
14          break;
15      case 4: Serial.println("A = 4");
16          break;
17      case 5: Serial.println("A = 5");
18          break;
19      case 6: Serial.println("A = 6");
20          break;
21      case 7: Serial.println("A = 7");
22          break;
23      case 8: Serial.println("A = 8");
24          break;
25      case 9: Serial.println("A = 9");
26          break;
27      case 10: Serial.println("A = 10");
28          break;
29      default: Serial.println("A is kleiner dan 1 of groter dan 10");
30    }
31  }
32
33  void loop() {
34    // leave empty for now
35  }
```

Om te zien hoe “default” werkt, zou je regel 6 kunnen aanpassen naar b.v. A = 11; waarbij alle “case” statements falen, en “switch” uit gaat komen bij het “default” statement. Mochten we echter “default” niet hebben geplaatst, dan zal er gewoon niets gebeuren omdat er geen geldig statement is gevonden.

Gebruik van “default” in een”switch” statement is optioneel …

Zoals gezegd, het “switch” statement wordt veel minder gebruikt dan het “if” statement. Overigens is het sterk aan te raden om de “case” waarden in een logische volgorde te zetten waardoor code beter leesbaar wordt en je geen onverwachte resultaten tegen gaat komen.

Probeer, indien mogelijk, om de casewaarden in een logische volgorde te plaatsen, voor betere leesbaarheid en ter voorkoming van onverwachte resultaten.

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 5: Lussen