6.0. Arduino Programmeren voor Beginners – Deel 6: Functies

In les 6 van Arduino Programmeren voor Beginners, gaan we kijken naar functies (Engels: functions). We hebben al een paar functies gebruikt en gezien, maar in dit deel gaan we kijken naar functies die we zelf kunnen maken.

Functies worden vaak gebruikt om een reeks stappen, welke herhaald of hergebruikt kunnen worden, samen te voegen naar een nieuwe instructie. Dit kan bijdragen aan efficiënter programmeren, maar ook aan een beter leesbaar programma.

Inhoudsopgave

6.1. Wat zijn Functies en Waarom hebben we ze nodig?

Als eerste hebben we theoretisch geen functies nodig voor een programma, echter onze code zou dan super lang worden om maar te zwijgen over de beroerde leesbaarheid van onze code. Daarnaast besparen functies ons een hoop werk – zeker als we de functie kunnen hergebruiken.

We hebben al een aantal functies gezien en we hebben er zelfs al twee verplichte functies gedefinieerd in onze Arduino programma's: "setup()" en "loop()".

Maar wat zijn dan die functies (functions)? En waarom willen we ze gebruiken?

Een functie (ook wel subroutine genoemd) kunnen we zien als een groep instructies, welke samen een specifieke taak uitvoeren.
We maken deze functies om ons programma beter georganiseerd te houden, beter leesbaar te maken, en vaak omdat we deze "taak" elders willen hergebruiken. En dat "elders" kan zelfs in een ander programma zijn.

In bepaalde talen noemt men een functie ook wel een "subroutine", en dat beschrijft precies wat het is: een soort klein programma in ons programma. En net als bij een programma, kan een functie ook functies gedefinieerd hebben in de functie – een functie is echter ook onderhevig aan de eerdergenoemde "scope". Dus een functie die in een functie gedefinieerd wordt is dus alleen maar bekend in de functie waar het gedefinieerd is – dus net zoals we zagen bij variabelen!

Ik kan me voorstellen dat dit net even te ver ging om te bevatten, het is iets waar we later vanzelf mee te maken gaan krijgen, hou het dus in gedachten.

We kunnen twee soorten functies maken: een functie die een waarde teruggeeft, en een functie die geen waarde terug geeft.
Die laatste is wat men, in sommige programmeertalen, een "procedure" noemt. De taal C echter, noemt beiden gewoon een functie, of in het Engels: een "function".

Een functie, is een groep instructies met een bepaalde taak in gedachten.

Wanneer definiëren we een functie?

Wanneer code meerder keren herhaald wordt in een programma,

Als onze code beter leesbaar wordt door het gebruik van functies,

Als we onze code beter kunnen beheren met functies

Als we onze code willen hergebruiken, b.v. in andere programma's.

Laten we eens een voorbeeld doorlopen zodat we het concept beter begrijpen.

Stel we hebben een hond, en we moeten die vier keer per dag uitlaten: Om 8:00, 12:00, 17:00, en om 21:00.

De taak "hond uitlaten" kan omschreven worden met de volgende instructies:
– Jas aandoen
– Hondenriem bij de hond aandoen

– Naar buiten gaan
– 15 minuten rondlopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.

Dus voor een hele dag zou onze “code” er zo uitzien:

Als het 8:00 is doe dan:
– Jas aandoen
– Hondenriem bij de hond aandoen
– Naar buiten gaan
– 15 minuten rondlopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.

Als het 12:00 is doe dan:
– Jas aandoen
– Hondenriem bij de hond aandoen
– Naar buiten gaan
– 15 minuten rondlopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.

Als het 17:00 is doe dan:
– Jas aandoen
– Hondenriem bij de hond aandoen
– Naar buiten gaan
– 15 minuten rondlopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.

Als het 21:00 is doe dan:
– Jas aandoen
– Hondenriem bij de hond aandoen
– Naar buiten gaan
– 15 minuten rondlopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.

Zo wordt ons programma best lang he? En we zien aardig wat herhaling van dezelfde instructies …

We zouden echter de functie “HondUitlaten()” kunnen definiëren, bijvoorbeeld:

HondUitlaten():
– Jas aandoen
– Hondenriem bij de hond aandoen

– Naar buiten gaan
– 15 minuten rond lopen
– Terug naar binnen gaan
– Hondenriem afdoen
– Jas uitdoen.

Nu wordt ons programma een stuk overzichtelijker:

```
Als het 8:00 is dan
  HondUitlaten()

Als het 12:00 is dan
  HondUitlaten()

Als het 17:00 is dan
  HondUitlaten()

Als het 21:00 is dan
  HondUitlaten()
```

Zoals je ziet is onze code nu een stuk netter en beter te lezen. We hebben nu de programmeertaal uitgebreid, voor ons programma, met de functie “HondUitlaten”.

In de meeste scenario’s, vooral als een functie meerdere keren gebruikt wordt in een programma, zal dit zelfs leiden tot een kleiner programma (na compileren) voor de Arduino of Computer. Het is efficiënter.

Maar functies hebben nog een groot voordeel. Stel we moeten de instructies voor het hond uitlaten aanpassen, bijvoorbeeld door “deur van het slot halen” ene “deur weer op slot doen” toe te voegen. Bij een functie hoeven we dat dus maar op 1 plaats te doen: in de functie definitie. In tegenstelling tot het eerste voorbeeld waar we dat op 4 plaatsen moeten veranderen. Onze code wordt dus beter beheersbaar.

Stel we hebben een meer generieke functie gemaakt, dan zouden we deze in een zogenaamde bibliotheek (librarys) kunnen zetten zodat we deze functie in andere programma’s ook weer kunnen gebruiken – en we dus tijd besparen wanneer we weer een ander programma gaan maken. Dit noemt met “re-use” of “hergebruik” van code. We gaan nog even niet in op het gebruik van libraries, dat is voor een stadium waarin we gevorderder zijn.

Vergeet niet:

Een functie is net als een klein programma … binnen een programma

Een functie kan zelf ook functies bevatten …

Een functie heeft een “scope” – zoals we zagen bij variabelen.

6.2. Zelf Functie(s) maken

In de programmeertaal C, zoals we die voor Arduino programmeren gebruiken, is gelukkig redelijk eenvoudig. De basisstructuur ziet er zo uit:

```
datatype FunctieNaam ( FunctieParameters ) {
  // functie code
}
```

Ik hoop dat je nog iets van DataTypes hebt onthouden zoals we dat in Deel 3 bespraken. Mocht dit niet zo zijn, dan kun je altijd nog even terug kijkenen ik zal een paar details hier weer vermelden.

De datatype geeft aan wat voor antwoord we terug mogen verwachten van de functie. Echter … niet alle datatypes zijn bruikbaar hiervoor, en dan moeten we vooral denken aan arrays!

De meest gebruikte datatypes voor functies zijn boolean, int, char, float en void (andere datatypen zoals double, long en de “unsigned” types werken ook).

Uiteraard moet onze functie ook een naam krijgen, de FunctieNaam, voor welke we dezelfde regels moeten volgens als namen voor variabelen en constanten:

Functie naam:

De Functie naam:

Moeten beginnen met een letter (a, b, …, z, A, B, … , Z) of een underscore (_)

Mag letters bevatten

Mag underscore(s) bevatten

Mag nummers bevatten

MAG GEEN speciale tekens, symbolen of spaties bevatten is hoofdletter gevoelig!

Een functie kan nul of meer Functie Parameters accepteren. Het aantal parameters is echter vast en voor gedefinieerd voor onze functie!

In een aantal C-programmeertaal dialecten, moeten we een functie declareren (aankondigen) voor we ze kunnen gebruiken. In C voor Arduino Programmeren is dit echter niet per se noodzakelijk en daarom gaan we daar hier niet op in.

We hebben overigens ook al een aantal functies gebruikt en aangeroepen, ook al waren we ons daar misschien niet van bewust, waarbij we parameters naar de functie sturen door ze tussen ronde haakje te zetten. Een voorbeeld: Serial.print("Hallo");

Hier roepen we de functie “print()” aan van object “Serial” en geven het de parameter “Hallo” (een string). Meer over objecten later.

De code block, tussen accolades en na de functie naam en parameters, bevat de instructies voor onze functie – net zoals we dat ook bij “if” en de verschillende loops zagen (“for”, “while” en “do … while …”).

Een simpele (en niet zo zinvolle) functie definitie zou kunnen zijn:

```
1void ZegHallo() {
2  Serial.println("Hallo");
3}
```

Deze functie, “ZegHallo()”, gebruikt geen parameters, omdat er niks tussen de ronde haakjes () staat. Het stuurt ook geen antwoord terug omdat we het datatype “void” gebruiken – wat zoveel betekend als “niks, nada, noppes”.

De “body” van de functie (de code blok), bevat de instructies van de functie, en in dit geval voert het alleen maar de tekst “Hallo” uit.

Een voorbeeld hoe we dit kunnen gebruiken:

```
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      ZegHallo();
7    }
8  }
9
10 void loop() {
11   // leave empty for now
12 }
13
14 void ZegHallo() {
15   Serial.println("Hallo");
16 }
```

Dit zou er een beetje vertrouwd uit moeten zien.

Aan het einde definieren we de functie “ZegHallo()”.
In de “for”-loop roepen we de functie 5 keer aan, wat zoiets als dit oplevert

```
Hallo
Hallo
```

Hallo
Hallo
Hallo

Makkelijk toch?

Waarden doorgeven aan een Functie

Dat laatste voorbeeld was natuurlijk bijzonder simpel. Laten we eens kijken hoe we de functie kunnen gebruiken in ons lampen voorbeeld door 5 lampen aan te zettenmet behulp van een “for”-loop.

Hiervoor maken we de functie “ZetLampAan”, welke we een lamp nummer gaan geven voor de lamp die aangezet moet worden.

We weten eigenlijk wel dat de lamp nummers van het datatype “int” gaat worden (zie ook de definitie van “A” in de “for”-loop).

We weten ook dat de functie niets terug hoeft te geven.

Al dat gecombineerd:

```
void ZetLampAan(int LampNummer) {
  Serial.print("Zet de volgende lamp AAN: ");
  Serial.println(LampNummer);
}
```

De parameter “LampNummer” is dus gedefinieerd als een “int”. Je ziet dat we de variabele “LampNummer” in onze functie definiëren. Het is belangrijk om te weten dat deze variabele de waarde aan gaat nemen van de parameter die we aan de functie doorgeven als we de functie aanroepen. Deze waarde wordt gekopieerd – dus als we de waarde van LampNummer veranderen, dan wordt de originele parameter (wat ook een variabele kan zijn) daardoor niet veranderd!

Om nog eens terug te komen op “scope”: de variabele “LampNummer” is natuurlijk niet bekend buiten de functie “ZetLampAan”.

Als we in regel 6, de functie aanroepen, geven we het de waarde van de variabele “A” mee. Deze waarde wordt de gekopieerd in de variabele “LampNummer”.

Alles even bij elkaar gezet:

```
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      ZetLampAan(A);
7    }
8  }
9
10 void loop() {
11   // leave empty for now
12 }
13
14 void ZetLampAan(int LampNummer) {
15   Serial.print("Zet de volgende lamp AAN: ");
16   Serial.println(LampNummer);
17 }
```

Zie je hoe de variabele “LampNummer” wordt gebruikt in de functie?

Dit is natuurlijk maar een eenvoudig voorbeeld. Wat gebeurt er nu als we meerdere waarden aan de functie door willen geven. Bijvoorbeeld door een boolean toe te voegen welke aangeeft of de lamp nu aan (true) of uit (false) gezet dient te worden?

Als we meerder parameters willen gebruiken, dan moeten we ze scheiden met komma’s (,).

Parameters in een functie worden gescheiden gehouden door een komma,
zowel bij functie definitie als bij functie aanroep.

Zoals met alle waarden die we door willen geven bij de parameters, moeten we ook die extra parameters voorzien van een naam (LichtAan) en een data type (boolean) zodat de functie weet wat voor waarde het kan verwachten.

Uiteraard moeten we even een “if” statement in de functie hangen, zodat afhankelijk van de boolean waarde, de lamp AAN of UIT gaat worden. Dit laat zien dat we functies zodanig kunnen schrijven dat ze voor meerdere situaties inzet baar kunnen zijn.

Het resultaat:

```
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      ZetLampAan(A, true);
7    }
8
9    for(int A=1; A<=5; A++) {
10     ZetLampAan(A, false);
11   }
12 }
13
14 void loop() {
15   // leave empty for now
16 }
17
18 void ZetLampAan(int LampNummer, boolean LampAan) {
19   if(LampAan) {
20     Serial.print("Zet de volgende lamp AAN: ");
21   }
22   else
23   {
24     Serial.print("Zet de volgende lamp UIT: ");
25   }
26
27   Serial.println(LampNummer);
28 }
```

Zoals je ziet de “for”-loop, gaat door 5 iteraties, waarbij 5 lampen AAN gezet gaan worden.
De doorgegeven waarden zijn dus ook weer apart gehouden door een komma te plaatsen!
In de volgende “for”-loop, weer 5 iteraties, waar we de lampen UIT zetten.

De output ziet er ongeveer zo uit:

Zet de volgende lamp AAN: 1
Zet de volgende lamp AAN: 2
Zet de volgende lamp AAN: 3
Zet de volgende lamp AAN: 4
Zet de volgende lamp AAN: 5
Zet de volgende lamp UIT: 1
Zet de volgende lamp UIT: 2
Zet de volgende lamp UIT: 3
Zet de volgende lamp UIT: 4
Zet de volgende lamp UIT: 5

Bedenk wel dat deze voorbeeld wel een beetje simpel zijn, maar zodra je jouw eigen programma's gaat schrijven, die wat groter zijn, dan zul je snel leren hoe handig functies kunnen zijn.

Antwoord van een Functie krijgen

We weten nu dus dat we een functiewaarden kunnen meegeven door middel van parameters. Maar hoe krijgen we een antwoord terug van een functie – bijvoorbeeld voor een complexe berekening?

Herinner je de "void" in onze voorgaande voorbeelden? Dat is waar we definiëren wat voor datatype we als antwoord mogen verwachten.

In de functie zelf moeten we echter duidelijk aangeven wat de "return" waarde is. De term "return" is Engels voor "terug", dus de waarde die we "terug" sturen.

Als we een functie definiëren welke een waarde terug gaat geven (dus niet "void"!) dan moeten we de instructie "return" gebruiken om een antwoord terug te sturen. Dat antwoord moet van hetzelfde datatype zijn als in de functie definitie!

We zouden bijvoorbeeld ons geld voorbeeld, dat we eerder hebben gebruikt, als voorbeeld kunnen gebruiken. Ook dit is natuurlijk geen ingewikkelde situatie, maar het dient slechts ter illustratie hoe een functie werkt.

Stel we maken een functie "TelAmMijnGeld", welke we twee parameters geven. Ik heb deze parameters expres een andere naam gegeven om aan te geven dat de waarden van de parameters gekopieerd worden. Maar … deze namen mogen gerust dezelfde zijn, want de waarde die als parameter wordt opgegeven is een ander variabele dan de variabele met dezelfde naam in de functie – denk terug aan de "scope" van variabelen.

In de functie dus 2 parameters, en als antwoord het "Totaal". De "return" waarde wordt vervolgens toegewezen aan de variabele "AlMijnGeld".

Probeer de volgende code maar eens.

```
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 = TelAmMijnGeld(ZakGeld, SpaarGeld);
14
15   // print the values to the serial monitor
16   Serial.print("ZakGeld = ");
17   Serial.println(ZakGeld);
18
19   Serial.print("SpaarGeld = ");
20   Serial.println(SpaarGeld);
21
22   Serial.print("AlMijnGeld = ");
23   Serial.println(AlMijnGeld);
24 }
25
26 void loop() {
27   // leave empty for now
28 }
29
30 int TelAmMijnGeld(int OpZak, int OpDeBank) {
31   int Totaal;
32
33   Totaal = OpZak + OpDeBank;
34
35   return Totaal;
36 }
```

Een functie die terugkomt met een waarde zouden we kunnen zien als een variabele welke een waarde bevat. Uiteraard kunnen we niets toewijzen aan deze zogenaamde variabele, maar we kunnen de waarde wel “uitlezen” en gebruiken waar we vaak ook een variabele of constante kunnen

gebruiken. Kijk in et volgende aangepaste voorbeeld maar eens naar regel 21 … We hebben als de “parameter” voor de functie “Serial.println() ” gewoon onze nieuwe functie gebruikt … en dat werkt prima! We hoeven dus de “return” (terug) waarde van een functie echt niet eerst in een variabele op te slaan – tenzij we deze waarde meerdere keren nodig hebben, want in zo’n geval willen we de berekening maar 1x doen, en iedere keer dat we de functie aanroepen, wordt de functie uitgevoerd.

Een functie welke een antwoord (return) waarde geeft kan gebruikt worden alsof het een variabele of constante is …

Elke keer als een functie wordt aangeroepen, zullen de instructies in de functie opnieuw uitgevoerd worden. Als een antwoord dus meerdere keren gebruikt wordt, overweeg dan om deze waarde in een variabele op te slaan in plaats van herhaaldelijk aanroepen van de functie.

```
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
9    // assign the values
10   ZakGeld = 4;
11   SpaarGeld = 12;
12
13   // print the values to the serial monitor
14   Serial.print("ZakGeld = ");
15   Serial.println(ZakGeld);
16
17   Serial.print("SpaarGeld = ");
18   Serial.println(SpaarGeld);
19
20   Serial.print("AlMijnGeld = ");
21   Serial.println( TelAlMijnGeld(ZakGeld, SpaarGeld) );
22 }
23
24 void loop() {
25   // leave empty for now
26 }
27
28 int TelAlMijnGeld(int OpZak, int OpDeBank) {
29   int Totaal;
30
31   Totaal = OpZak + OpDeBank;
```

32	
33	return Totaal;
34	}

Ik adviseer weer om wat te gaan spelen met functies, bedenk een eigen functie, voeg wat extra parameters toe, etc.

6.3. Functies die Zichzelf aanroepen (Recursie)

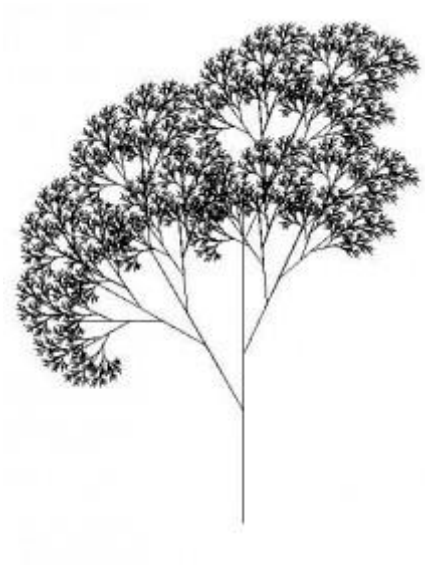
Deze paragraaf is wat lastig te bevatten en zeker niet super belangrijk – sla dit dus gerust over als het onduidelijk wordt.

Een functie kan zichzelf aanroepen, en men noemt dit “Recursie“.

Je denkt vast dat je nu aangekomen bent bij het gekkenhuis, en ik kan me voorstellen dat dit onderwerp een beetje te complex is voor de meeste beginners en zelfs ervaren programmeurs hebben soms moeite met dit onderwerp. Je hebt recursie zelden nodig en ik vermeld het hier alleen maar om een beetje meer volledig te zijn.

Voor wie interesse heeft, lees gerust verder, als je even niet meer snapt waar dit over gaat: ga gerust naar Deel 7.

Recursie is een zeer krachtig, ook al is het soms moeilijk te bevatten, hulpmiddel voor programmeurs, waarmee je met weinig code toch iets moois voor elkaar kunt krijgen. Hieronder twee resultaten: een recursief getekende boom en de zogenaamde zeef van Sierpinski (een driehoek in een driehoek in een driehoek, etc). Afbeeldingen bron: Wikipedia – Tree door Brentsmith101, Sierpinski door Wereon.



Recursieve Boom

Zeef of Driehoek van Sierpinski

OK, laten we een makkelijker voorbeeld bedenken om recursie uit te leggen.

Stel we hebben de nummers 1 tot en met 5 en we willen ze allemaal optellen, dus 1+2+3+4+5 (= 15). We kunnen dit heel simpel doen natuurlijk, met een eenvoudig “for”-loop, maar vandaag gaan we lekker moeilijk doen en gaan we recursie gebruiken.

We gaan een functie maken die we het nummer “5” geven, welke daarbij “4” gaat optellen. Daarna geven we dezelfde functie het nummer “4” en tellen daar “3” bij op, etc.

```
1  int VoegVorigeNummerToe(int Nummer){
2    if(Nummer==0)
3      return Nummer;
4    else
5      return Nummer+VoegVorigeNummerToe(Nummer-1);
6  }
```

Wat deze functie doe is de waarde van “Nummer” pakken en kijken of dit nul is. Als het nul is, stuur dan nul terug (return). Als dit niet nul is, roep dan de functie “VoegVorigNummerToe” aan met “Nummer – 1”.

Maf he?

In andere woorden: deze functie begint met een nummer, en blijft er steeds het voorgaande nummer aan toevoegen tot de nul bereikt heeft.

Laten we dit eens van dichtbij gaan bekijken met een voorbeeld:

```
1  void setup() {
2    // set the speed for the serial monitor:
3    Serial.begin(9600);
4
5    Serial.println(VoegVorigeNummerToe(5));
6  }
7
8  void loop() {
9    // leave empty for now
10 }
11
```

```
12  int VoegVorigeNummerToe(int Nummer){
13    if(Nummer==0)
14      return Nummer;
15    else
16      return Nummer+VoegVorigeNummerToe(Nummer-1);
17  }
```

We starten dus met VoegVorigeNummerToe(5).

De waarde van “Nummer” is niet nul, dus we doen 5 + “VoegVorigeNummerToe(5-1)”. We hebben nog geen return waarde of antwoord omdat eerst de aanroep van VoegVorigeNummerToe(5-1) afgewerkt moet worden.
In deze tweede aanroep van “VoegVorigeNummerToe” is “Nummer” gelijk aan 4, en dus nog steeds niet nul. We doen dus 4 + “VoegVorigeNummerToe(4-1)” en weer moeten we eerst de aanroep van “VoegVorigeNummerToe(4-1)” eerst afwerken voor we een antwoord terug kunnen sturen.
In de derde aanroep is “Nummer” gelijk aan 3 en dus weer niet nul, en we doen dus weer 3 + “VoegVorigeNummerToe(3-1)” – aan.
In de vierde aanroep is “Nummer” gelijk aan 2 en dus weer niet nul, en we doen dus weer 2 + “VoegVorigeNummerToe(2-1)” – aan.
In de vijfde aanroep is “Nummer” gelijk aan 1 en dus weer niet nul, en we doen dus weer 1 + “VoegVorigeNummerToe(1-1)” – aan.
De zesde aanroep echter is “Nummer” wel gelijk aan nul en we sturen dus nul als antwoord terug.

Omdat we nu in de scope van de vijfde aanroep terugkomen, en we dus een return waarde kunnen bepalen, gaan we weer een stap terug met het antwoord (1+0 = 1).
Vervolgens komen we terug in de scope van de vierde aanroep, met als resultaat 2+1 = 3.
Dit herhaald zich voor de scope van de derde functie aanroep, met als resultaat 3+3 = 6,
en de scope van de tweede aanroep met als resultaat 4+6 = 10,
en de scope van de eerste aanroep wat uiteindelijk het antwoord 5+10 = 15 levert.

Lekker verwarrend he?

Misschien helpt het als ik het zo laat zien:

1ste aanroep: VoegVorigeNummerToe(5) // in de setup() functie
2de aanroep: VoegVorigeNummerToe(4) // in de VoegVorigeNummerToe(5) functie aanroep
3de aanroep: VoegVorigeNummerToe(3) // in de VoegVorigeNummerToe(4) functie aanroep
4de aanroep: VoegVorigeNummerToe(2) // in de VoegVorigeNummerToe(3) functie aanroep
5de aanroep: VoegVorigeNummerToe(1) // in de VoegVorigeNummerToe(2) functie aanroep
6de aanroep: VoegVorigeNummerToe(0) // in de VoegVorigeNummerToe(1) functie aanroep, Nummer is nu NUL!
terug naar de 5de aanroep: Return = Nummer + resultaat van de 6de aanroep = 1 + 0 = 1
terug naar de 4de aanroep: Return = Nummer + resultaat van de 5de aanroep = 2 + 1 = 3
terug naar de 3de aanroep: Return = Nummer + resultaat van de 4de aanroep = 3 + 3 = 6
terug naar de 2de aanroep: Return = Nummer + resultaat van de 3de aanroep = 4 + 6 = 10
terug naar de 1ste aanroep: Return = Nummer + resultaat van de 2de aanroep = 5 + 10 = 15

Eindresultaat: = 15

Ik weet dat dit concept moeilijk te bevatten is, zelfs voor ervaren programmeurs, omdat dit niet de gebruikelijk manier is zoals onze grijze cellen hier mee omgaan. Je hoeft alleen maar te onthouden dat de functie zichzelf kan aanroepen en zichzelf daarbij niet in de weg zit, maar dat iedere aanroep een eigen “scope” heeft.

Als vuistregel voor recursie: zorg er altijd voor dat de functie een “uitgang” heeft, anders blijft de functie eindeloos zichzelf aanroepen wat als gevolg heeft dat de computer of de Arduino geen geheugen meer heeft en vast loopt. Iedere scope wordt namelijk opgeslagen voor wanneer de functie weer “terug” komt …

Bij Recursief gebruik van functies:

ALTIJD zorgen dat er een “uitgang” is voor de functie!

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 7: Strings