Verslag les -5- 2-4-19

Werkende lussen: for-loop; while-loop; do… while- loop.

Wanneer iets twee of meerdere keren gedaan moet worden, dan gebruiken we een subroutine, loop, functie. (verschillende woorden voor hetzelfde begrip)

De FOR lus:

```
void setup() {                  // terug naar waar je vandaan kwam
  Serial.begin(115200);         // communicatie snelheid met de PC
  int A = 6;                    // de variabele
  for(A=1; A<=5; A++) {         // hier wordt opgeteld ++ is: 1 erbij en bij:  -- 1 er af.
    Serial.print("Schakel deze lamp AAN: ");  // stukje tekst
    Serial.println(A);          // de verschillende cijfers
   }
}
void loop() {
 // even leeg laten
}
```

Dit geeft als resultaat:
```
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
```

Voorbeeld aftellen:

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

Geeft als resultaat:
```
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
Schakel deze lamp AAN: 0
```

Je kan en mag ook de variabele binnen de lus definiëren:

```
 for(int A=1; A<=5; A++) {    // ← de variabele wordt hier gedeclareerd en gebruikt.
    Serial.print("Schakel deze lamp AAN: ");
    Serial.println(A);
   }                            // ← na deze accolade is de scoop van A afgelopen!
Serial.print("Schakel deze lamp AAN: ");
// Serial.println(A);   ← hier gaat het fout!! De variabele A is niet gedeclareerd.
}
```

Bovenstaande lus geeft hetzelfde als in de eerste lus. (deze dus)
Voor een teller wordt vaak de letter I of N gebruikt.

De Schets (gedeelte)	Het resultaat:
`Serial.println("Optellen:");`	Optellen:

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

`Serial.println("Aftellen:");`	Schakel deze lamp AAN: 5

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

Ook een loop kan je met een **break** verlaten!
Bijvoorbeeld:

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

De While lus

Veronderstel, een lamp die aan moet als het donker wordt, 'szomers is dat anders dan in de winter:

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

Zolang de stelling 'DonkerBuiten' waar is, staat de lamp aan.
Uiteraard is hiervoor een sensor nodig, maar dat komt verderop wel aan de orde, bovendien zal de lamp met een enkele tekstregel niet gaan branden, wat wel nodig is, komt ook later.

Als een while loop 'false' oplevert, wordt hij niet uitgevoerd!

De do….while lus

Bij een do….while lus kan je dus ook negatieve situaties aan.

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

Een voorbeeld:

```
void setup() {
  Serial.begin(115200);
  int A = 1;
  while(A<=5) {
    Serial.println("WHILE: A is nog steeds <= 5");
    A = A + 1;
  }
  A = 1;
```

```
  do {
    Serial.println("DO WHILE: A is nog steeds <= 5");
    A = A + 1;
  } while (A<=5);
}

void loop() {
  // laat dit voorlopig leeg
}
```

Het eindresultaat is:
```
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
```

Als we **A=1** veranderen in int **A=10**, dan krijgen we: **DO WHILE: A is nog steeds <= 5** en dat terwijl A al die tijd groter dan 5 is!!
Bij een do-while lus wordt de lus altijd minimaal één keer doorlopen en uitgevoerd!
Omdat pas na afloop van het doorlopen van de lus, op waarheid wordt gecontroleerd.

Je bent niet verplicht om met cijfers te werken, een start en stop variabele mag ook

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

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

De opdracht: **Dobbelsteen**

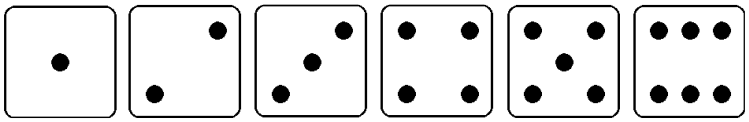
Louis had in les -4- onderstaande oefening opgegeven.

Een dobbelsteen simuleren met een Arduino kan op diverse manieren: je kunt het resultaat van een worp op een LCD display laten zien, of via een 7-segments LED schermpje tonen, maar in dit verhaal gaan we uit van afzonderlijke LEDjes, die de ‘ogen’ van de dobbelsteen moeten voorstellen.

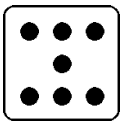
Omschrijving van het project:
Laat, na het drukken op een knop, de uitkomst van een ‘worp’ met een dobbelsteen, door middel van LEDjes zien, in een patroon dat overeenkomt met de ‘ogen’ op een dobbelsteen.

Analyse van het project
Voor elk project geldt: voordat we in het wilde weg aan het breadboard prikken slaan is het wenselijk een degelijke analyse van het project te maken. Ook in dit ogenschijnlijk simpele geval zal blijken dat een nadere beschouwing ons zal helpen bij het maken van de sketch.

Laten we allereerst een dobbelsteen eens onder de loep nemen.
De ‘ogen’ zijn als volgt verdeeld:



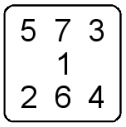
De bedoeling is dat we deze 6 ‘patronen door middel van LEDjes gaan weergeven.
Als eerste merken we op dat er 7 plaatsen zijn waar een ‘oog’ gezet wordt, n.l.:



Dit houdt in dat we 7 LEDjes nodig zullen hebben, die we op zo’n manier op het breadboard plaatsen, dat het patroon hiernaast benaderd wordt.

Laten we deze 7 ‘ogen’ als volgt nummeren, Dan kunnen we het volgende concluderen:

- Oog 1 komt alleen in de oneven uitkomsten voor.
- Oog 2 en oog 3 komen altijd samen voor.
- Oog 4 en oog 5 komen altijd samen voor.
- Oog 6 en oog 7 komen altijd samen voor.

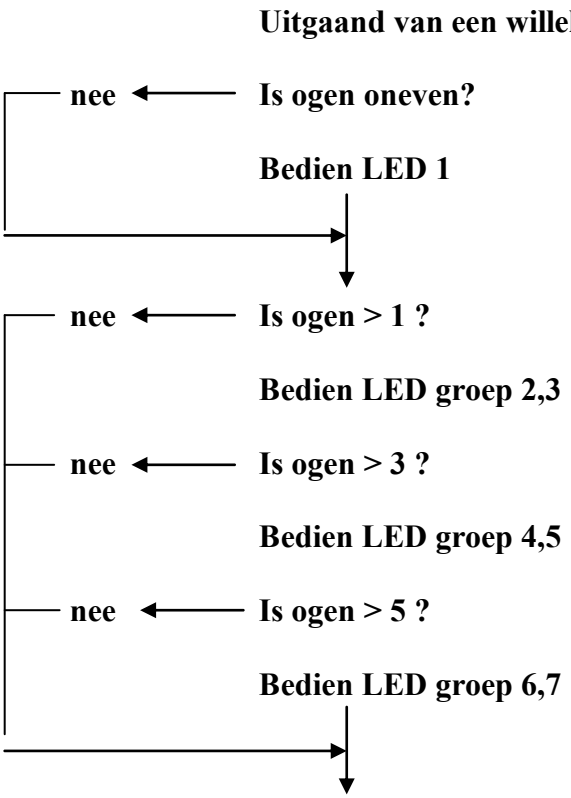


Dit houdt in dat we te maken hebben met 4 groepjes van LEDjes in plaats van 7 afzonderlijke LEDjes! 2 en 3 kunnen we parallel (of in serie) schakelen, dat geldt ook voor 4 en 5, & 6 en 7. Zo kunnen we 2 LEDjes in één opdracht bedienen (ze zitten beide op dezelfde pin).

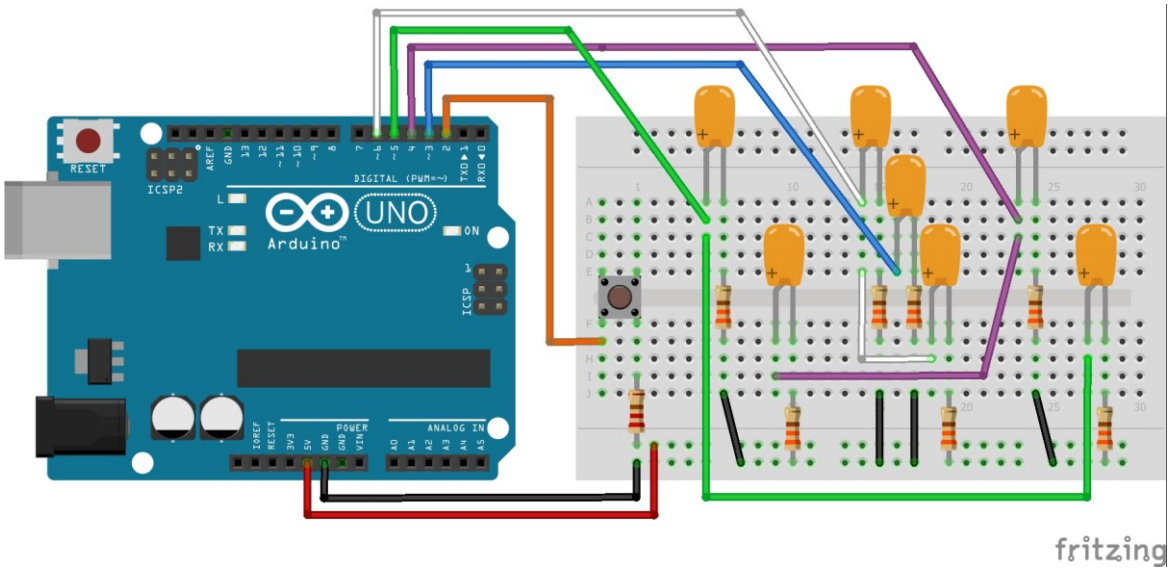
Er is nog iets opmerkelijks.

- 2 en 3 komen altijd voor als de worp groter is dan 1.
- 4 en 5 komen altijd voor als de worp groter is dan 3.
- 6 en 7 komen altijd voor als de worp groter is dan 5.

Dit alles maakt onze sketch wel zeer eenvoudig.
Het ‘hart’ van de sketch, waar een willekeurig geheel getal tussen 1 en 6 wordt omgezet in het bedienen van de overeenkomstige LEDjes, ziet er schematisch als volgt uit:



Ik heb met opzet ‘bedienen’ van de LEDjes gebruikt, omdat dit zowel het aan- als uitzetten van LEDjes omvat. Bovenstaand schema kan dus in een subroutine gebruikt worden, waarin het aan- of uitzetten als parameter voorkomt.
Op de club zullen we de sketch bespreken, maar probeer hem eerst zelf eens te maken..



Ter verfraaiing:

- Simuleer het ‘rollen’ van de dobbelsteen.
- Zet na een bepaalde tijd, dat er niet op de knop gedrukt wordt, (b.v 10 seconden) alle LEDjes uit.

De oplossing:

Er waren diverse oplossingen, de een nog mooier dan de andere.
Louis had onderstaande fraaie oplossing. Wat mij betreft is het toch heel knap opgelost!

```
const int knop=2;                  // knopje op pootje -2- aangesloten
const int led1=3;                  // centrale LEDje op potje -3-
const int led23=4;                 // LEDjes 3 en 3 op pootje -4-
const int led45=5;                 // LEDjes 4 en 5 op pootje -5-
const int led67=6;                 // LEDjes 6 en 7 op pootje -6-
int ogen=7;                        // variabele, het aantal geworpen ogen. Waarom op 7? Komt verderop!
unsigned long vorigeKnopGedrukt = 0; // tijdsverloop voor het uitschakelen na 10 sec geen actie.
boolean uitgezet = true;           // true als de LEDjes uit zijn.

void setup() {
 randomSeed(millis);               // basis voor het maken van een willekeurig getal. Aanvankelijk werd
                                   // randomSeed(analogRead(0)); gebruikt. Het ruis signaal op de
                                   // analoge ingang. Maar millis werkt ook goed, hierbij blijft het
                                   // analoge pootje vrij voor andere toepassingen.
 pinMode(knop,INPUT);              // pootje voor het knopje (2) is ingang
 pinMode(led1,OUTPUT);             // pootjes voor de diverse LEDjes zijn uitgang
 pinMode(led23,OUTPUT);
 pinMode(led45,OUTPUT);
 pinMode(led67,OUTPUT);
}

void loop() {
 if(uitgezet==false){                          // als de LEDjes aan staan:
   if(millis()-vorigeKnopGedrukt>10000){       // na 10 sec verloopt deze tijd.
     leds(7,LOW);                              // alle LEDjes gaan uit (uitleg -7- komt in de functie)
     uitgezet=true;                            // hier worden alle LEDjes uitgeschakeld.
                                               // 7 is oneven, dus -1- is uit, 7> 1, 3 en 5 dus alles uit.
```

```
    }
   }
   if(digitalRead(knop)==HIGH){ // als de knop wordt ingedrukt, gaat pootje -2- high
    for(int i=3;i<20;i++){      // gedurende korte tijd rolt de dobbelsteen steeds langzamer.
     leds(ogen,LOW);            // de worp (het aantal ogen) wordt verderop in de subroutine uitgezet
     delay(50);                 // korte pause
     ogen=random(1,7);          // een willekeurig getal tussen 1 en 6 wordt gemaakt.
     leds(ogen,HIGH);           // de ogen worden overeenkomstig aan gezet. Zie functie/subroutine.
     delay(i*40);               // pauze tussen twee waarden wordt langer, het uitrollen vertraagd.
    }
    vorigeKnopGedrukt=millis(); // nu start de 10 sec wachttijd. Na 10 sec is de worp onzichtbaar.
    uitgezet=false;             // LEDjes staan aan.
   }
  }

  void leds(int ogen,int aanUit){   // het schakelen van de ledjes. De worp is gedaan, het getal is bekend!
   if(ogen % 2 == 1){               // (% = modulo, zie les -3-) De rest is 1 bij elke oneven worp.
    digitalWrite(led1, aanUit);     // dan moet het centrale LEDje (nummer -1-) altijd aan.
                                    // bij het uitrollen van de dobbelsteen is bepaald welke LEDjes aan of
                                    // uit worden geschakeld, immers NA het uitschakelen wordt de
   }                                // nieuwe worp gedaan. (ogen=random(1,7);)
   if(ogen>1){                      // het kan 2, 3, 4, 5 of 6 zijn.
    digitalWrite(led23,aanUit);     // schakel groep 23 aan, bij >1 zijn altijd deze twee leds aan.
    if(ogen>3){                     // de worp kan ook 4, 5 of 6 zijn.
     digitalWrite(led45,aanUit);    // dan ook groep 45 aanzetten, bij >3 zijn groepen 23 en 45 altijd aan.
     if(ogen>5){                    // als het 6 is, dan is groep 67 altijd aan.
      digitalWrite(led67,aanUit);   // groep 67 aan zetten, bij -6- is een even getal, dus LED -1- is uit.
     }
    }
   }
  }
```

Wat als er -7- geworpen wordt? Nou, dat gebeurd niet!
Er bestaat geen zevenzijdige dobbelsteen.
Bij de start is het aantal ogen 7. Alle LEDjes worden uitgeschakeld.
In de eerste vergelijking wordt gekeken of het getal oneven is, 7 is oneven, waardoor LED 1 wordt uitgeschakeld. Omdat 7 > dan 1, 3 en 5, worden alle LEDjes uitgeschakeld.
Verderop is deze -7- niet meer nodig, want dan is er altijd een oude worp bekend.

Nadat de LEDjes overeenkomstig met de worp zijn ingeschakeld, worden ze nadien weer uitgeschakeld. Bij het uitschakelen is de oude worp nog steeds geldig, er wordt pas opnieuw geworpen **nadat** de LEDjes zijn uitgeschakeld.

Uiteraard kan ik mijn oplossing ook hieronder zetten, net als die van anderen, maar bovenstaande oplossing is toch wel héél fraai!

Rob v. Dijk is iets geavanceerder bezig geweest, hij had een heel kort programma omdat hij rechtstreeks de registers van de Arduino bewerkte. Dat is inderdaad slimmer en sneller, maar gaat toch iets boven de insteek van deze cursus.

Voor de nieuwsgierigen onder ons:

```
#define interruptPin 2
volatile bool gooiStatus = false;

void setup() {
  DDRB = B00111111;                 // zet PORTB (digitale pinnen 13~8) naar OUTPUT
  randomSeed(analogRead(0));        // genereert een nieuwe seed voor elke gooi
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), start, FALLING);
}

void loop() {
  if (gooiStatus == true) {
    PORTB = B00000000;              // zet digitale pinnen 13-8 naar LOW
    delay(200);
    byte willekeurigGetal;          // definieert geheel getal van 0 tot en met 255
    willekeurigGetal = random(1, 7); // genereert een getal tussen 0 en 7 (1,2,3,4,5 of 6)
    switch (willekeurigGetal) {
      case 1:
        PORTB = B00000001;          // zet pin 8 naar HIGH
        break;
      case 2:
        PORTB = B00000011;          // zet pin 8 en 9 naar HIGH
        break;
      case 3:
        PORTB = B00000111;          // zet pin 8,9 en 10 naar HIGH
        break;
      case 4:
        PORTB = B00001111;          // zet pin 8,9,10 en 11 naar HIGH
        break;
      case 5:
        PORTB = B00011111;          // zet pin 8,9,10,11 en 12 naar HIGH
        break;
      case 6:
        PORTB = B00111111;          // zet pin 8,9,10,11,12 en 13 naar HIGH
      default:
        break;
    }
    gooiStatus = false;
  }
}

void start() {
  if (gooiStatus == !true ) {
    gooiStatus = true;
  }
}
```