Verslag les 6 9-4-19 **Functies**

Functies (subroutines) maken het programma overzichtelijker en sneller.
Twee functies die altijd aanwezig moeten zijn, zijn: **setup()** en **loop()**.
Altijd met twee haakjes, waar wel of geen argument in kan staan.

Een functie kan in een andere functie worden gedefinieerd, een functie heeft ook een scoop. (les -3-) nesten kan, maar de geldigheid blijft, net als bij variabelen, beperkt binnen dit nest.
Er zijn twee typen functies: -1- functies die een waarde terug geven
-2- functies die GEEN waarde terug geeft. (procedure)

Wil je een functie in een ANDER programma ook kunnen gebruiken, dan zet je hem in de bibliotheek (library). Dit komt later in de cursus aan de orde.

De algemene opbouw van een functie:

```
datatype FunctieNaam (functie parameters) {   // DataTypes zie les -3-
  // de functie code
}
```

De meest gebruikte datatypen voor functies zijn: boolean, integer, character, float en void.
De functienaam moet met een letter of underscore _ beginnen!

Je mag parameters meegeven, maar die liggen meteen vast bij het definiëren van de functie.
Een functie hoef je in Arduino niet vooraf te declareren.
Een functie voorbeeld: **Serial.print ("Hallo");**
Een functie die met het data type **void** begint, geeft NIETS terug, void betekent: 'niets'.
De haakjes zijn WEL verplicht, dat geldt voor elke functie, zoals ook bij. **void setup() {**
Een onzinnig, maar werkend voorbeeld:

```
void setup() {
  Serial.begin (115200);
 for(int i=1; i<=5;i++){          // de functie moet 5 keer worden uitgevoerd.
    ZegHallo();                   // hier wordt de functie: ZegHallo aangeroepen
  }
}

void loop() {                     // deze functie is altijd verplicht aanwezig!
  // put your main code here, to run repeatedly:
}
void ZegHallo(){                  // het begin van de functie die niets terug geeft.
  Serial.println("hallo");        // de taak: printen op het scherm.
}
```

Een voorbeeld waarbij data wordt meegegeven naar de functie:

```
void setup() {
 Serial.begin(115200);
 for(int A=1; A<=5; A++) {
    ZetLampAan(A);                        // hier wordt de waarde A meegegeven
  }
}

void loop() {                             // laat dit nog even leeg.
}

void ZetLampAan(int LampNummer) { // je mag de data hier een andere naam geven
                                  // de scoop van de naam blijft geldig binnen de functie.
```

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

Het resultaat:
```
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
```

Wil je méér argumenten meegeven, dan moet je ze met een komma **,** scheiden.
Je kan ook booleaanse data meegeven zoals true en false

```
void setup() {
  Serial.begin(115200);
  for(int A=1; A<=5; A++) {
    ZetLampAan(A, true);
  }
  for(int A=1; A<=5; A++) {
    ZetLampAan(A, false);
  }
}

void loop() {
  // laat dit nog leeg
}

void ZetLampAan(int LampNummer, boolean LampAan) {
  if(LampAan) {
    Serial.print("Zet de volgende lamp AAN: ");
  }
  else
  {
    Serial.print("Zet de volgende lamp UIT: ");
  }
  Serial.println(LampNummer);
}
```

Lampen worden eerst allemaal aan gezet, daarna weer uitgeschakeld.

Data van een functie terug krijgen

*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 gebruiken hiervoor de eerder toegepaste geld situatie, in de functie: TelAlMijnGeld.

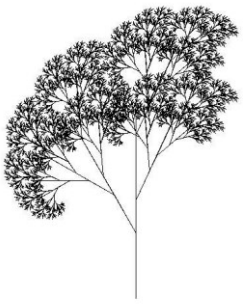
```
void setup() {
 Serial.begin(115200);
  int ZakGeld;
  int SpaarGeld;
  int AlMijnGeld;
  ZakGeld = 4;
  SpaarGeld = 12;
  AlMijnGeld = TelAlMijnGeld (ZakGeld, SpaarGeld);
                // de functie aanroep met 2 argumenten, die aldaar worden opgeteld.
```

```
  Serial.print("ZakGeld = ");
  Serial.println(ZakGeld);
  Serial.print("SpaarGeld = ");
  Serial.println(SpaarGeld);
  Serial.print("AlMijnGeld = ");
  Serial.println(AlMijnGeld);
 }
 void loop() {
  // laat dit nog leeg
 }
// Hieronder de functie, die integer is, en met twee (nieuwe!) integer variabelen werkt.
  int TelAlMijnGeld (int OpZak, int OpDeBank) {  // geef wel de typen variabelen aan
   int Totaal; // de variabelen moeten van hetzelfde type zijn als in het hoofdprogramma
   Totaal = OpZak + OpDeBank; // je mag hier zelf andere namen kiezen (scoop lokaal)
    return Totaal;  // return geeft aan dat er een waarde wordt mee gegeven.
// De waarde krijgt in het hoofdprogramma de naam die in de functie aanroep is vermeld.
// Hier is dat: AlMijnGeld. Dat is integer, omdat de functie integer is. (int TelAlMijnGeld).
 }
```

Hier gebruik je het 'toverwoord' **return** waarmee wordt aangegeven dat je een waarde meegeeft bij de terugkeer naar het hoofdprogramma.
Worden er meer waarden terug gegeven, dan ook hier met een komma scheiden.

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.

Functies die zichzelf aanroepen (recursie)



Dit komt niet zo vaak voor, maar voor de volledigheid toch vermeldt.
Hiernaast een tekening die met recursie ie getekend, allemaal driehoeken
Het begint met driehoeken:

Uiteindelijk heb je de boom (fractals?)

Stel, je wilt de cijfers 1 + 2 + 3 + 3 + 4 + 5 optellen, dat kan eenvoudig met een for- lus.
We doen we het op de moeilijke manier: met recursie.
Je geeft het getal -5- en de boel loopt! De functie roept zichzelf opnieuw op, en telt 4 erbij op.

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

Wat deze functie doet 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”.

```
void setup() {
 Serial.begin(115200);
 Serial.println(VoegVorigeNummerToe(5)); // de functie wordt aangeroepen
                                         // hierbij wordt het getal -5- meegegeven.
}
void loop() {
  // leave empty for now
}
int VoegVorigeNummerToe(int Nummer){ // opnieuw functie aanroepen
  if(Nummer==0)                      // dit blijft doorgaan tot nummer == 0
     return Nummer;                  // als de voorwaarde klaar is, volgt return.
   else
     return Nummer+VoegVorigeNummerToe(Nummer-1);
}                                    // zolang nummer niet 0 is, opnieuw.
```

Het kan ook zijn dat je zo in een oneindige lus terecht komt. Dit gaat door totdat het geheugen vol is en de boel vast loopt. Elk nest heeft een eigen scoop, die wordt opgeslagen.
Zorg er voor dat er altijd een 'uitgang' is.

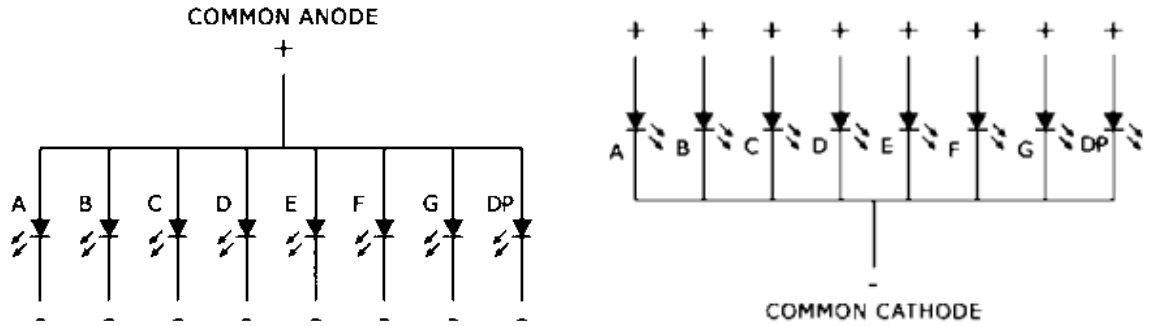
-=-=-=-

Dobbelsteen met 7 segments display

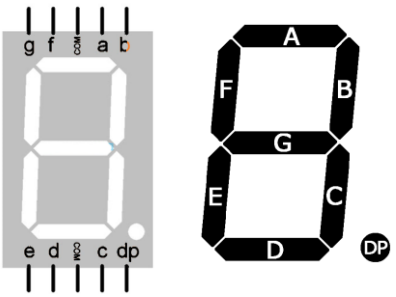
Door Louisvan Duuren.

Om het resultaat van een worp met een dobbelsteen weer te geven, volstaat een enkelvoudig 7 segments LCD display. Er bestaan 2 types:

1. Common Anode (in het vervolg CA genoemd)
2. Common Cathode (in het vervolg CC genoemd)



We zullen als voorbeeld het CA type nemen, ook mede omdat ik alleen maar over dit type beschik, maar met wat kleine aanpassingen geldt dit verhaal ook voor een CC type. Ik zal dat, waar het van belang is, vermelden.
De 7 segmentjes worden met de letters A tot G weergegeven, de decimale punt met DP.
De corresponderende aansluitpennen zijn zoals hiernaast:



In onderstaande tabel staat weergegeven hoe de cijfers o t/m 9 met behulp van de 7 segmentjes worden weergegeven. 1 is aan en 0 is uit.
De eerste regel met aanduiding 'niets', (alles uit) is er bijgezet om als optie te dienen om het display uit te zetten.
Omdat we met een dobbelsteen te maken hebben gaat de tabel voor dit geval maar t/m de regel met '6'. (de eerste 7 regels.)

#	a	b	c	d	e	f	g
niets	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0

#	a	b	c	d	e	f	g
niets	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1

Ook de decimale punt (DP) hoeven we niet te gebruiken en komt daarom ook niet voor in de tabel. We hebben dus te maken met 7 segmenten, die op 7 digitale pinnen op de Arduino moeten worden aangesloten en 7 verschillende weergaven op het display (niets en de cijfers 1 t/m 6). Dit vraagt om het werken met **arrays**!

De tabel kunnen we in een 2 dimensionale array, genaamd segmPerCyfer[][], plaatsen:

```
const int segmPerCyfer[7][7]={ {0,0,0,0,0,0,0},   // alles uit
                               {0,1,1,0,0,0,0},   // 1
                               {1,1,0,1,1,0,1},   // 2
                               {1,1,1,1,0,0,1},   // 3
                               {0,1,1,0,0,1,1},   // 4
                               {1,0,1,1,0,1,1},   // 5
                               {1,0,1,1,1,1,1}    // 6
                              };
```

De eerste [7] is het aantal weer te geven karakters (niets, 1 t/m 6) (het aantal rijen).
De tweede [7] is het aantal segmenten per karakter (het aantal kolommen).
En de 7 digitale aansluitpinnen op de Arduino plaatsen we als volgt in een 1 dimensionale array onder de naam pin[]: **const int pin[7]={2,3,4,5,6,7,8};**

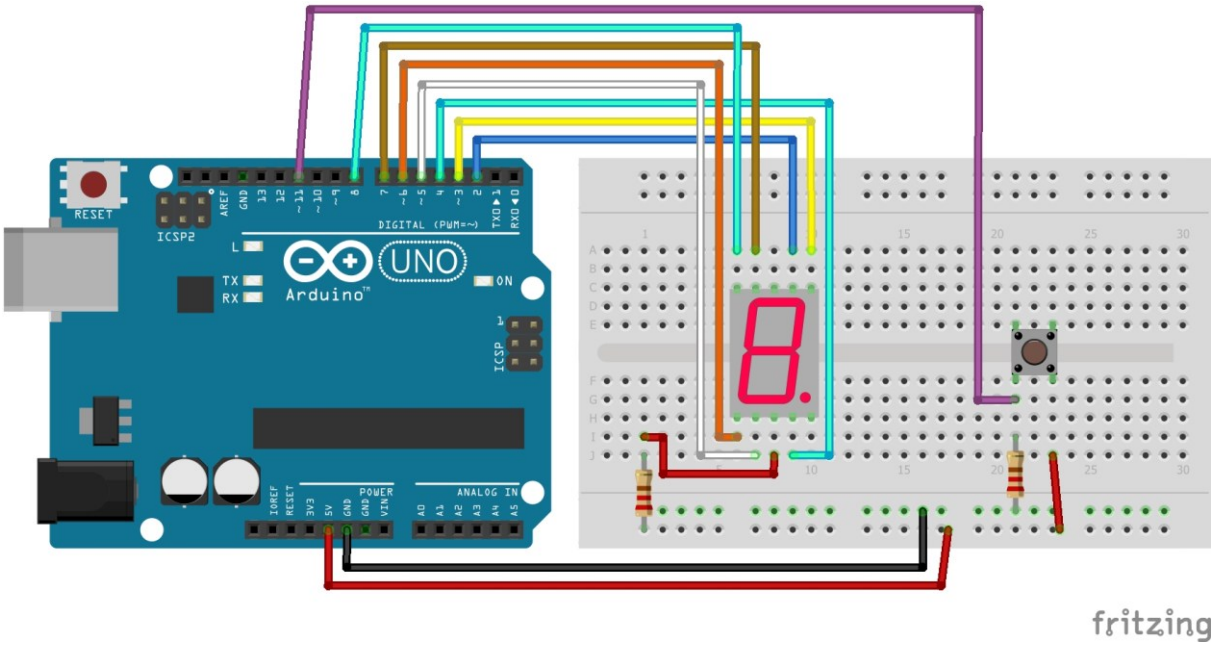
We zien dat we bij het declareren van een array het werkelijke aantal elementen moeten opgeven. Bij het benaderen van (werken met) de afzonderlijke elementen moeten we er rekening mee houden dat ze geïndexeerd worden vanaf **0 ←!**

segmPerCyfer[3][1] b.v. verwijst naar het cijfer 3, elementje b (een 1, dus aan).
Arrays in combinatie met for opdrachten, vormen een zeer efficiënte manier van programmeren, met:

```
for(int i=0;i<7;i++){
    pinMode(pin[i],OUTPUT); // zet je 7 pinnen op OUTPUT.
}
```

De inhoud van de segmPerCyfer[][] array geldt, zoals hij hier is gebruikt, voor CC displays. CA display bezitters zouden de nullen in enen en de enen in nullen kunnen veranderen, er is een elegantere manier om dit te bereiken, n.l. gebruik maken van logische operator **!** (not).

Hieronder de bedrading.

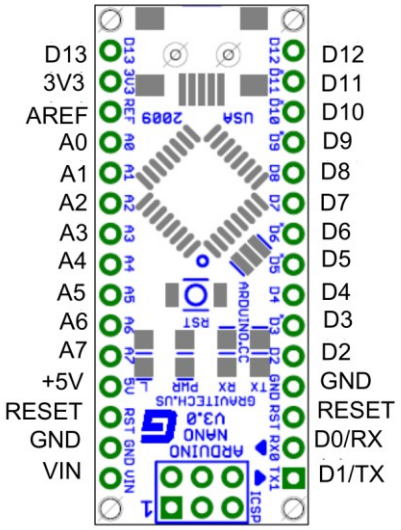


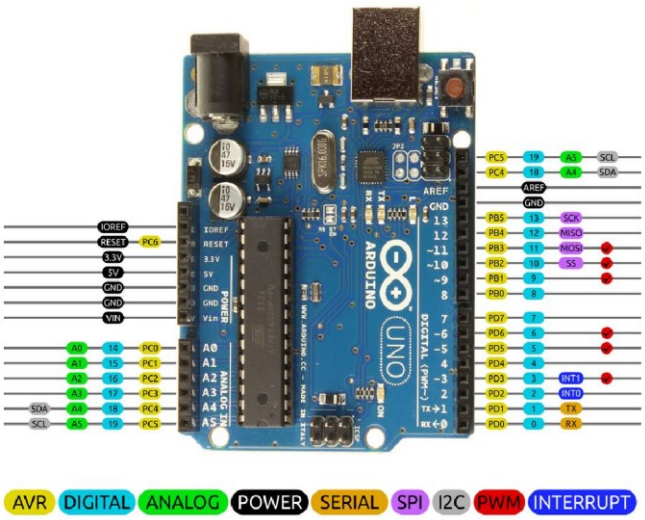

```
// dobbelsteen met 7 segment LED display
// de decimale punt op het display wordt niet gebruikt
// gebaseerd op Common Anode display
// LED status (0 =uit, 1 = aan) voor de cijfers 1 t/m 6 en alles ui (wis)

const byte segmPerCyfer[7][7]={{0,0,0,0,0,0,0}, // alles uit
                               {0,1,1,0,0,0,0},  // 1
                               {1,1,0,1,1,0,1},  // 2
                               {1,1,1,1,0,0,1},  // 3
                               {0,1,1,0,0,1,1},  // 4
                               {1,0,1,1,0,1,1},  // 5
                               {1,0,1,1,1,1,1}}; // 6
const byte pin[7]={2,3,4,5,6,7,8};          // LED pootjes a,b,c,d,e,f,g
const byte knop = 11;                       // knop op pin 11
unsigned long tijdVorigeKnop = 0;           // tijd sinds laatste knop bediening.
boolean uitgezet=true;                      // vlag voor gewist of niet
byte ogen;                                  // aantal 'gegooide' ogen (1 - 6)

void setup() {
  randomSeed(analogRead(0)); // nodig om de random() functie willekeurig te starten

 for(byte i=0;i<7;i++){
    pinMode(pin[i],OUTPUT);                 // zet alle 7 LED pinnen op OUTPUT
  }
  pinMode(knop,INPUT);
  leds(0);                  // roept de leds() routine aan met 0 = alle LEDjes uit (wis)
}

void loop() {
  if(uitgezet==false){                      // als nog niet uitgezet (gewist)
    if(millis()-tijdVorigeKnop>10000){      // als 10 seconden zijn verstreken
      leds(0);                              // zet alle 7 ledjes uit (wis)
      uitgezet=true;                        // zet vlag uitgezet aan
    }
```

```
  }
  if(digitalRead(knop)==HIGH){          // is de knop ingedrukt
   for(byte i=1;i<20;i++){          // 'rol' met de dobbelsteen (19 willekeurige waarden)
    leds(0);                              // zet alle 7 LEDjes uit (wis)
    delay(50);
    ogen=random(1,7);                     // genereer willekeurig getal tussen 1 en 6
    leds(ogen);         // roep de routine leds() aan met het aantal ogen als argument
    delay(i*30);
   }
   tijdVorigeKnop=millis();         // stel het tijdstip in waarop de knop is ingedrukt
   uitgezet=false;                       // zet de vlag 'uitgezet' op niet waar
  }
 }
 void leds(byte ogen){     // maak het LED patroon overeenkomstug het argument 'ogen'
  for(byte j=0;j<7;j++){
   digitalWrite(pin[j],!segmPerCyfer[ogen][j]);
                                  // de NOT (!) is nodig voor Common Anode diplays
                                  // laat de NOT (!) weg voor Common Cathode displays
  }
 }
```

Achteraf blijkt dat het begrip 'array' pas in les -8- wordt behandeld.
Bewaar deze dobbelsteen dus voor na deze les.

Hieronder de pootjes van Nano en Uno



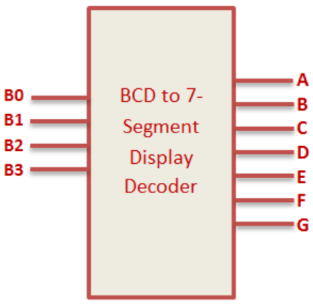


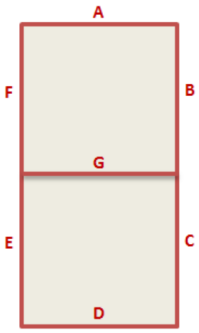
De 7-segments dobbelsteen van Rob v. Dijk

```
const int ledgroep1  = 4;  // led 1
const int ledgroep2  = 6;  // led 2 en 3
const int ledgroep3  = 3;  // led 4 en 5
const int ledgroep4  = 5;  // led 6 en 7
const int knop = 2;
void setup() {
 pinMode(ledgroep1,OUTPUT);
```

```
  pinMode(ledgroep2,OUTPUT);
  pinMode(ledgroep3,OUTPUT);
  pinMode(ledgroep4,OUTPUT);
  pinMode(knop,INPUT);
  Serial.begin(9600);
  randomSeed(analogRead(0));//Om het toevalsproces van de functie random te starten
}
void loop()
{
int value;
if (digitalRead(knop) == HIGH) { // Ja! De schakelaar is ingedrukt.
value = random(1, 7); //Kies een getal van 1 t/m 6
if(value==1)     digitalWrite(ledgroep1,HIGH);
if(value==2) {    digitalWrite(ledgroep2,HIGH);
}
if(value==3) {    digitalWrite(ledgroep1,HIGH);
                  digitalWrite(ledgroep2,HIGH);
}
if(value==4) {    digitalWrite(ledgroep2,HIGH);
                  digitalWrite(ledgroep3,HIGH);
}
if(value==5) {    digitalWrite(ledgroep1,HIGH);
                  digitalWrite(ledgroep2,HIGH);
                  digitalWrite(ledgroep3,HIGH);
}
if(value==6) {    digitalWrite(ledgroep2,HIGH);
                  digitalWrite(ledgroep3,HIGH);
                  digitalWrite(ledgroep4,HIGH);
}
delay(1000);
digitalWrite(ledgroep1,LOW);
digitalWrite(ledgroep2,LOW);
digitalWrite(ledgroep3,LOW);
digitalWrite(ledgroep4,LOW);
}
else {
      digitalWrite(ledgroep1,HIGH);
      delay(500);
      digitalWrite(ledgroep1,LOW);
      digitalWrite(ledgroep2,HIGH);
      delay(500);
      digitalWrite(ledgroep2,LOW);
      digitalWrite(ledgroep3,HIGH);
      delay(500);
      digitalWrite(ledgroep3,LOW);
      digitalWrite(ledgroep4,HIGH);
      delay(500);
      digitalWrite(ledgroep4,LOW);
}
}
```

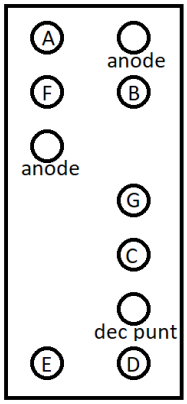
7 segments display van Dré

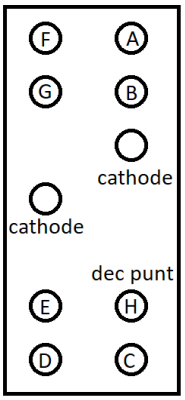


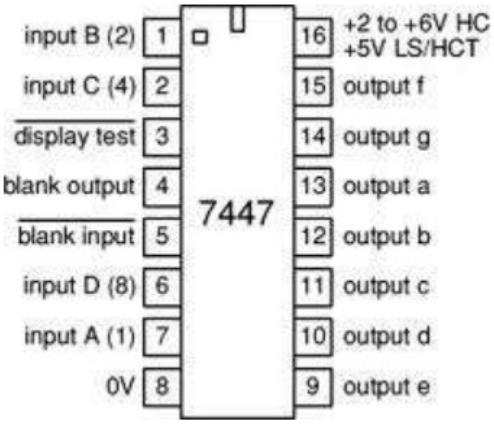


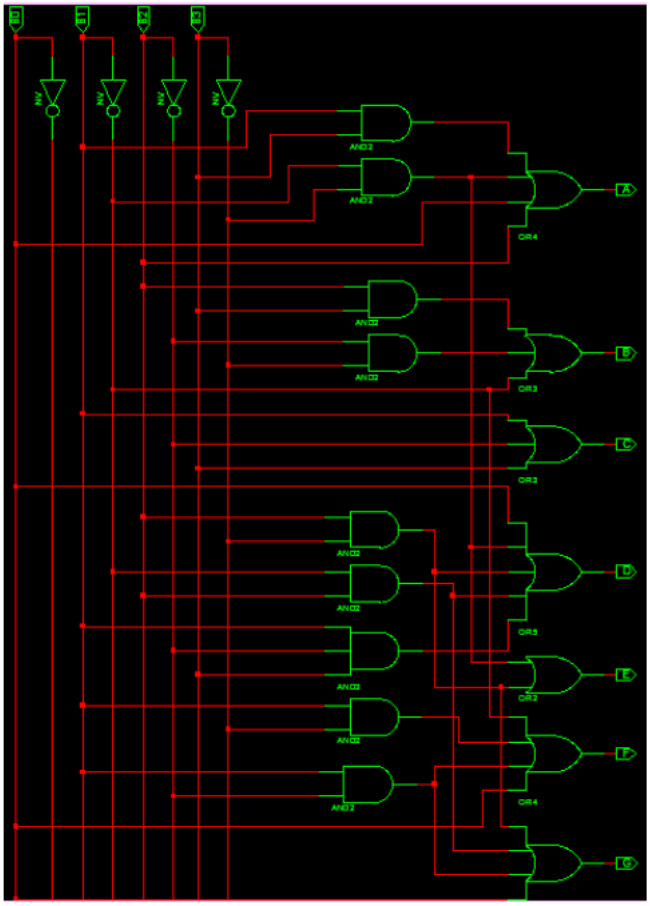
waarheids tabel:

B3 B2 B1 B0	A B C D E F G
0000	0000001
0001	1001111
0010	0010010
0011	0000110
0100	1001100
0101	0100100
0110	0100000
0111	0001111
1000	0000000
1001	0000100









0 = ABCDEF
1 = BC
2 = ABDEG
3 = ABCDG
4 = BCFG
5 = ACDFG
6 = ACDEFG
7 = ABC
8 = ABCDEFG
9 = ABCDFG

Er zijn twee varianten display's een CA en CC exemplaar. De pinning verschilt onderling ook nog eens. Sterker nog, het door Louis gebruikte display heeft een wéér andere pinning.

De CA 'draait' op een encoder chip, wordt met BCD signaal aangestuurd
De CC wordt rechtstreeks uit de Arduino aangestuurd.
Ik heb het 'dobbelsteen' verhaal gelaten voor wat het was, de displays tellen van 0……9 etc.

```
// declaratie
  byte a=0;                          // de variabele

void setup() {
  DDRB = B00001111;                  // zet D0...3 uitgang
  PORTB = B00000000;                 // zet pootjes laag
  DDRD = B01111111;                  // zet D8....14 uitgang
  PORTD = B00000000;                 // zet pootjes laag
}

void loop() {
  for (a=0; a<=9; a++) {             // tellertje van 0 ….9
    PORTB=(a);                       // poort B wordt in één keer geset
    if(a==0){PORTD=(B00111111);}     // 0 = ABCDEF
    if(a==1){PORTD=(B00000110);}     // 1 = BC
    if(a==2){PORTD=(B01011011);}     // 2 = ABDEG
    if(a==3){PORTD=(B01001111);}     // 3 = ABCDG
    if(a==4){PORTD=(B01100110);}     // 4 = BCFG
    if(a==5){PORTD=(B01101101);}     // 5 = ACDFG
    if(a==6){PORTD=(B01111101);}     // 6 = ACDEFG
    if(a==7){PORTD=(B00000111);}     // 7 = ABC
    if(a==8){PORTD=(B01111111);}     // 8 = ABCDEFG
    if(a==9){PORTD=(B01101111);}     // 9 = ABCDFG
    delay(1000);                     // 1 sec pauze
  }
}
```

Het display bevat 7 LEDs die óf met de kathode dan wel met de anode met elkaar zijn verbonden. De common kathode verbindt je met de **-** en de common anode aan de +.
Als je **KNAP** bent, kan je dat makkelijk onthouden **K**athode **N**egatief, **A**node **P**ositief.

Het toeval wil dat ik twee verschillende typen displays heb, beide typen heb ik aangesloten.
Er bestaan chips waarin al het 'denkwerk' is ingebouwd (7447) je stuurt het met 0.....9 aan.
Hiervoor gebruik je 4 pootjes, heb ik de digitale pootjes 8-9-10-11 voor gebruikt.
Dat andere type display heb ik rechtstreeks op de digitale pootjes 0-1-2-3-4-5-6 aangesloten.

De aansturing: het blijkt in de Arduino mogelijk om rechtstreeks de registers aan te sturen.
De pootjes 0..7 heten samen poort D. De pootjes 8...13 vormen samen poort B.
Er zijn registers waarin de pootjes ingang, of uitgang kunnen zijn: **D**ata **D**irection **R**egister.
In de setup maak ik van de betreffende pootjes uitgangen, in het programma stuur ik de hele poort in één keer aan. Zo laat ik beide displays hetzelfde weergeven: de cijfers 0.....9.
De een met chip, de andere rechtstreeks uit Arduino.
Aan het aansturen van registers kleven wat wat risico's, er kunnen 'vreemde dingen' gebeuren.
Dus ook dit is iets dat later aan de orde zal komen, valt buiten deze cursus.

Dobbel gerust nog even verder, maar dan op naar les -7- !

Voorgestelde opdracht voor komende week:
Een kolom van 8 leds, waarbij aan de hand van een potmeter meer of minder leds aan gaan.
Dit is een thermometer schaal.
Ook één ledje dat hoger en lager in de kolom verschuift, dat is een spot-schaal.
En dan nog onafhankelijk van de potmeter, net als knight rider, een 'heen en weer' gaande led.

Groeten, Dré

Nog even verder dobbelen. Hier nog wat mogelijkheden.
Als je tijd over hebt, dit is zeer interessant, maar nu heb ik er even geen tijd voor.

```
//  Dobbelsteen met 7-segment display
/*  Voor het aansturen van een 7-segments display gebruiken we een 2-dimensinele array,
    waarin  we vastleggen welke segmenten (a t/m g plus dp = decimale punt) voor welke
    getallen (0 t/m 9) moeten worden aangezet.

    Voor het dobbelsteen project gebruiken we geen decimale punt en maar 6 (1 t/m 6) van de
    10 getallen.
    Een 6x7 array volstaat dus in dit geval; in de 1ste dimensie de 6 getallen en in de 2e
    dimensie de 7 segmenten.
    Omdat in de array uitsluitend de gehele getallen 0 en 1 voorkomen, volstaat het datatype
    byte voor de array definitie. (eigenlijk is dit een enorme verspilling van geheugenruimte
    omdat slechts 1 bit van elke byte (= 12.5%) wordt gebruikt.
    Arduino C kent echter geen kleinere datatypes.
    Oplossingen hiervoor zijn mogelijk door toepassing van bit manipulatie).

    De toegepaste array is bruikbaar voor zowel CommonCathode als CommonAnode
    displays. Een "1" staat voor aansturing van een segment.

    In dit programma is gekozen voor een sequentiële aansturing van de segmenten. De reden
    Is dat gelijktijdige aansturing door de parallelschakeling van de segmenten een hogere
    belasting voor de Arduino betekent. Immers max 6 segmenten betekent, afhankelijk van de
    voorschakelweerstand, zo'n 6x20 mA = 120 mA aan stroomsterkte. Kan de Arduino in dit
    geval gemakkelijk aan, maar wanneer meerdere stroom vragende aansluitingen worden
    toegepast kan dat een probleem worden.
    Beter is het daarom om op elk moment slechts 1 segment aan te sturen. Dan is de belasting
    altijd slechts 20 mA. De methode die gebruikt wordt is om in FOR-lus te toetsen welke
    segmenten van het gekozen getal moeten worden aangestuurd (de segmenten met een '1" in
    de array). Die segmenten laat in hoog temp knipperen (1 microseconde voor zowel aan als
    uit). Als een segment niet hoeft te worden aangestuurd dan blijft dat segment in het ELSE-
    deel Uit. De frequentie is zo hoog dat het oog dat niet kan volgen; ze de getallen lijken
    continue aangestuurd te worden.

    Er is een softwarematige oplossing toegevoegd om het effect van contact-dendering door
    de gebruikte knop uit te schakelen. Zonder die oplossing werden bij het indrukken van de
    knop meerdere willekeurige getallen gegenereerd.
    De oplossing is heel eenvoudig. Er wordt als de knop wordt ingedrukt, eerst een
    debouncefunctie gestart, die een instelbare debounceTijd wacht voordat de startfunctie het
    willekeurige getal tussen 0 en 5 gaat genereren.

    De startfunctie genereert een getal tussen 0 en 5 om uit te array de getallen 1 t/m 6 te
    kunnen selecteren.
*/

#define knopPin 2

const byte nummers[6][7] = {  // a, b, c, d, e, f, g  display segmenten
  {0, 1, 1, 0, 0, 0, 0}, // getal 1
  {1, 1, 0, 1, 1, 0, 1}, // getal 2
  {1, 1, 1, 1, 0, 0, 1}, // getal 3
```

```
  {0, 1, 1, 0, 0, 1, 1}, // getal 4
  {1, 0, 1, 1, 0, 1, 1}, // getal 5
  {1, 0, 1, 1, 1, 1, 1}, // getal 6
};

const byte pinNr[7] = {6, 7, 8, 9, 10, 11, 12};
byte willekeurigGetal;
bool laatsteKnopStatus = LOW;
bool huidigeKnopStatus = LOW;

void setup() {
  for (int i = 0; i < 7; i++) {
    pinMode(pinNr[i], OUTPUT);
  }
  pinMode(knopPin, INPUT_PULLUP);
}

void loop() {
  huidigeKnopStatus = debounce(laatsteKnopStatus);
  if (laatsteKnopStatus == LOW && huidigeKnopStatus == HIGH) {
    start();
  }
  laatsteKnopStatus = huidigeKnopStatus;
 for (byte i = 0; i < 7; i++) {  // Serial.print(nummers[willekeurigGetal][i]);
    if (1 == nummers[willekeurigGetal][i]) {
      digitalWrite(pinNr[i], HIGH);
      delay(1);
      digitalWrite(pinNr[i], LOW);
      delay(1);
    }
    else
    {
      digitalWrite(pinNr[i], HIGH);
      delay(2);
    }
  }
}

bool debounce(bool laatsteStatus) {
  bool huidigeStatus = digitalRead(knopPin);
  if (laatsteStatus != huidigeStatus) {
    delay(5);
    huidigeStatus = digitalRead(knopPin);
  }
  return huidigeStatus;
}

void start() {
  randomSeed(analogRead(0));
  willekeurigGetal = random(0, 6); // genereert een getal (0,1,2,3,4,of 5)
}

// -=-=-=-=-=-=-=-
```

```
#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;
  }
}
```