HO
GENT

Deel 1: Arduino kennismaking

Wat is een microcontroller, structuur van een programma, syntax,…

HO GENT

Wat is een microcontroller

HO
GENT

Wat is een microcontroller?

Microcontroller = kleine “dedicated” computer.

- Beperkt in mogelijkheden (geheugen, snelheid,…)
- Voorzien van I/O voor sturing
- Alomtegenwoordig
- Één toepassing
- Honderden families, veel familieleden.
  (zie Wikipedia)
- Programmeren high level – low level

HO
GENT

Arduino

Is een verzamelnaam voor een ontwikkelplatform en dus geen microcontroller.

Komt in verschillende versies:

- UNO : originele versie (ATmega168, later ATmega 328p)
- Leonardo : ATmega32U4
- Lillypad: ATmega168V of ATmega328V (low power)
- …

HO
GENT

Structuur en syntax

HO
GENT

Programma

- C-taal
- Opgebouwd uit functies
- Twee basisfuncties: setup() en loop()

```
void setup()
{
	statements;
}
void loop()
{
	statements;
}
```

HO GENT

Wat is een functie?

[Type teruggegeven waarde] functienaam ([parameters]) { code; }

Voorbeeld:

```
void setup()
{
}
```

Mag ook:

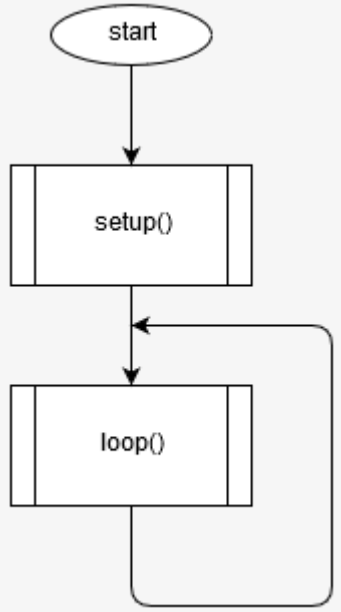
```
void setup() {}
```

HO GENT

basisstructuur

Terug naar basisstructuur van een Arduino programma:

```
void setup()
{
	statements;
}
void loop()
{
	statements;
}
```



Void (=leegte) : betekent dat deze functie geen enkele waarde teruggeeft.

HO GENT

Voorbeeldprogramma: Blink.ino

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

HO
GENT

Syntax

Accolades: { } geven begin en einde van een programmacodeblok aan.

Dit kan in een functie

```
void loop() {
    statements;
}
```

Maar ook in een instructie zoals een for-instructie

```
for (int i=0;i<10;i++) {
    statements;
}
```

puntkomma: ; is scheidingsteken tussen twee of meer statements.

```
    digitalWrite(13, HIGH);
    delay(1000);
```

HO GENT

commentaar

/*…*/ blok commentaar. Alles tussen /* en */ wordt genegeerd tijdens compilatie.

// regel commentaar. Alles na // op deze regel wordt genegeerd tijdens compilatie.

HO
GENT

variabelen

HO
GENT

Wat is jouw type?

Een variabele wordt gekenmerkt door zijn type, zijn waarde en zijn scope (bereik). Het type van de variabele bepaalt welke waarden er kunnen worden in opgeslagen, en hoe de waarde precies opgeslagen wordt.

Byte: 8 bit (=1 byte). Bereik van 0-255.

Int: 16-bit (=2 byte).
- Standaard signed: bereik -32.768 tot +32.767
- unsigned: bereik 0 tot 65.537

Long: 32-bit (=4 byte).
- standaard signed: bereik -2.147.483.648 tot 2.147.483.647
- unsigned: bereik 0 tot 4.294.967.295

Float: 32-bit (=4 byte). Zie single precision floating point (digitale elektronica)

Double: 64-bit (=8 byte). Zie double precision floating point (digitale elektronica)

Opmerking: float en double bewerkingen: traag!!!

HO GENT

Type (vervolg)

Char: 8 bits (=1 byte). Bevat ASCII code van een karakter. Opgepast: char is signed, dus bereik van -128 tot +127. Toekenning gebruikt single quotes of numerieke waarde.

Char k = ‘A’;
Char k = 65;

Beide toekenningen zijn evenwaardig.

String: variabele lengte, afhankelijk van hoeveel tekens. Toekenning met double quotes. Null-terminated, dus 1 byte meer dan aantal letters.

Opgepast: type string <> klasse String. (zie later).

Arrays: reeks van waarden van hetzelfde type. Zero-indexed: eerste element [0].

int myArray[] = {waarde0, waarde1, waarde2...}

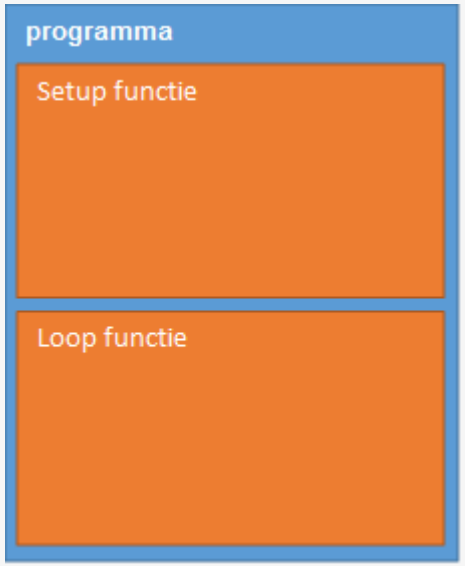
myArray[2] = 17;

HO GENT

Scope?

Een variabele is gedefinieerd binnen een bepaalde scope (bereik) van het programma. Het bestaan van de variabele beperkt zich tot deze scope. Erbuiten bestaat deze variabele niet.
De scope is het codeblok waarbinnen de variabele gedefinieerd werd. We kunnen hierbij het boxmodel gebruiken:

Het programma = hoofdbox
Elke functie = subbox
Elk instructie = subbox



HO GENT


Bestand Bewerken Schets Hulpmiddelen Help
seg7_blinkA §
void setup() {
const int segA = 4;
pinMode(segA,OUTPUT);
}
void loop() {
digitalWrite(segA,HIGH);
delay(500);
digitalWrite(segA,LOW);
delay(500);
}
'segA' was not declared in this scope
Foutmeldingen kopiëren
exit status 1
'segA' was not declared in this scope
1
Arduino Leonardo on COM7

Opgelet: 2 verschillende variabelen met zelfde naam binnen het programma.

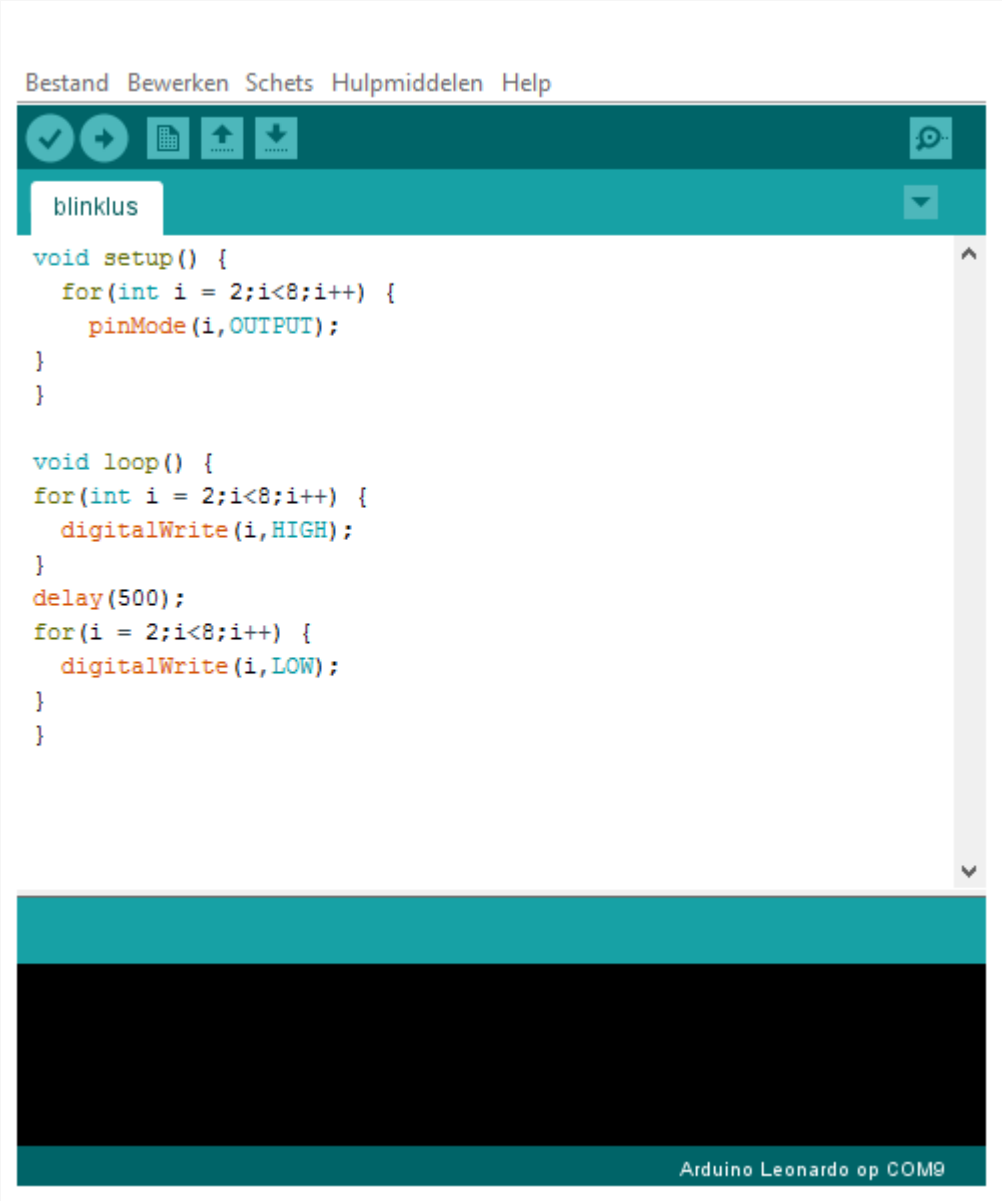
De ene segA heeft waarde 5 binnen setup().

De andere segA heeft waarde 4 binnen de loop().

Nog erger:

SegA met waarde 5 gedefinieerd over volledig programma, dus ook in functie loop().

Binnen loop() echter 2^{e} segA gedefinieerd met waarde 4. Deze lokale variabele heeft voorrang.



Waar komt hier een foutmelding?

bewerkingen

HO
GENT

Type van resultaat

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

Resultaat afhankelijk van type.
Bv.:

```
int a,b;
a=9;
b=2;
println(a/b);
```

Geeft 4 i.p.v. 4.5 omdat beide variabelen integers zijn.

Bewerking met 2 verschillende types: resultaat krijgt grootste type.

Opgepast voor overflow:

```
byte a = 250;
a=a+50;
```

Kan niet meer in de byte.

HO GENT

Samengestelde bewerkingen:

```
x++        // is hetzelfde als x = x + 1, of verhoog x met +1
x--        // is hetzelfde als as x = x - 1, of verlaag x met -1
x+= y      // is hetzelfde als as x = x + y, of verhoog x met +y
x-= y      // is hetzelfde als x = x - y, or of verlaag x met -y
x*= y      // is hetzelfde als x = x * y, of vermenigvuldig x met y
x/= y      // is hetzelfde als x = x / y, of deel x met y
```

Vergelijkingen:

```
x == y     // x is gelijk aan y
x != y     // x is niet gelijk aan y
x < y      // x is kleiner dan y
x > y      // x is groter dan y
x <= y     // x is kleiner of gelijk aan y
x >= y     // x is groter of gelijk aan y
```

HO GENT

Logische bewerkingen:

Logische AND:
if (x > 0 && x < 5) // waar alleen als beide vergelijkingen waar zijn

Logische OR:
if (x > 0 || y > 0) // waar als één van de twee vergelijkingen waar zijn

Logische NOT:
if (!x > 0) // alleen waar als vergelijking niet waar is

Bitwise bewerkingen:

&	(bitwise and)
|	(bitwise or)
^	(bitwise xor)
~	(bitwise not)
<<	(bitshift left)
>>	(bitshift right)

HO GENT

constanten

HO
GENT

Constante?

Een constante is voor Arduino hetzelfde als een variabele, alleen verandert de waarde nooit.
Er zijn enkele voorgedefinieerde constanten (altijd hoofdletters):

Booleaans:
- TRUE (!=0) en FALSE (=0)
- HIGH en LOW

Pindefinitie:
- HIGH en LOW

Zelf constanten definiëren:

```
const float pi = 3.14;
```

Alternatief: werken met #define (niet aangeraden!)

```
#define pi 3.14
```

HO GENT

programmastructuren

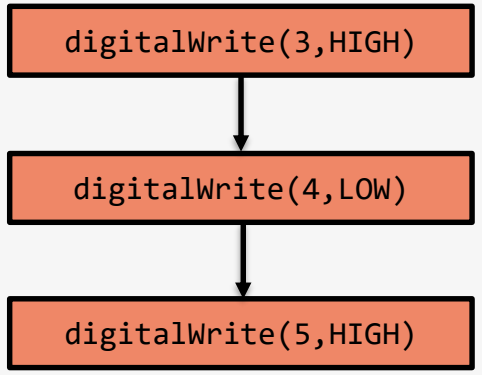
HO
GENT

Sequentie:

De instructies worden in een blok na elkaar geplaatst, gescheiden door puntkomma.

Een programmablok wordt geplaatst tussen accolades.

```
{
    digitalWrite(3,HIGH);
    digitalWrite(4,LOW);
    digitalWrite(5,HIGH);
}
```

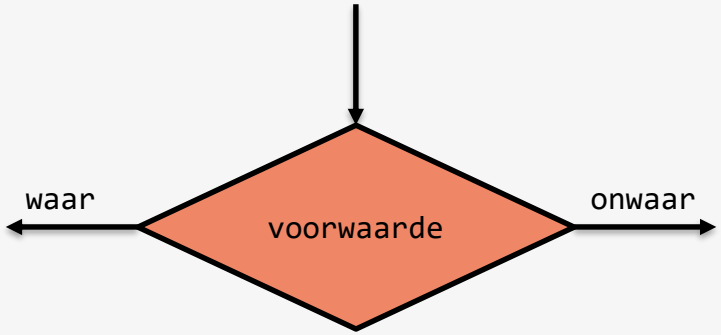


HO GENT

Selectie:

Een codeblok wordt uitgevoerd indien voldaan wordt aan een voorwaarde

```
If() {
    statements indien waar; }
else {
    statements indien onwaar; }
```



HO GENT

Selectie:

Een codeblok wordt uitgevoerd indien voldaan wordt aan een voorwaarde

```
switch (var) {
    case 1:
      //do something when var equals 1
      break;
    case 2:
      //do something when var equals 2
      break;
    default:
      // if nothing else matches, do the default
      // default is optional
    break;
  }
```

HO
GENT

Iteratie:

Een codeblok wordt herhaald (zolang voldaan wordt aan een voorwaarde)

```
for (variabele; conditie; expressie)
{
doeiets;
}
```

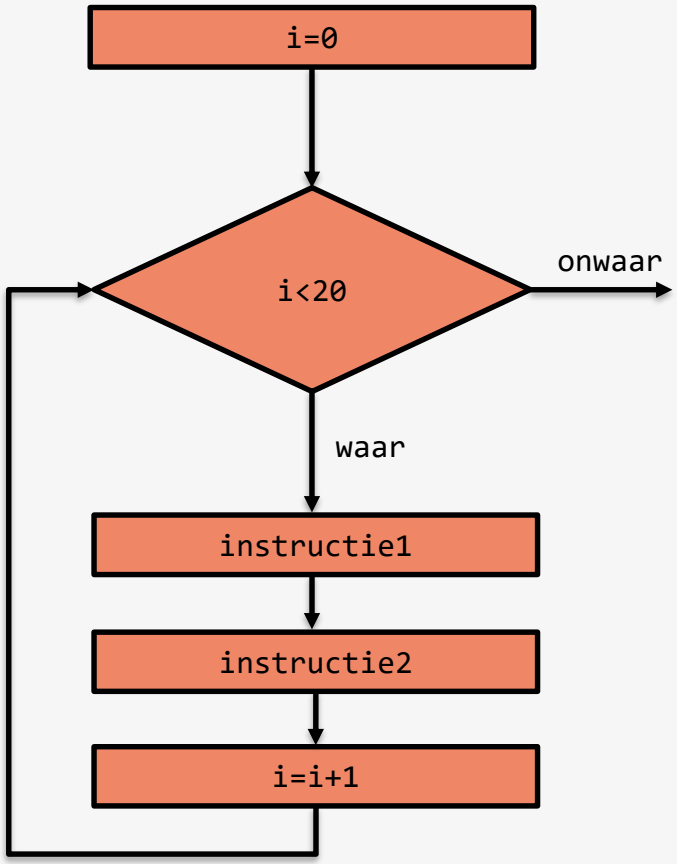
Voorbeeld

```
for (int i=0; i<20; i++)     // declareer i, en test of het kleiner is
{                            // dan 20, I wordt met 1 opgehoogd
    digitalWrite(13, HIGH);  // zet pin 13 aan
    delay(250);              // 1/4 second pauze
    digitalWrite(13, LOW);   // zet pin 13 uit
    delay(250);              // 1/4 second pauze
}
```

HO GENT

iteratie

```
for (int i=0; i<20; i++) {
     instructie1;
     instructie2;
}
```



HO GENT

Iteratie:

```
while (someVariable ?? value)
{
    doe iets;
}
```

Doe iets zolang aan de voorwaarde voldaan is.

```
do
{
    doe iets;
} while (someVariable ?? value);
```

Idem, maar doordat de voorwaarde onderaan staat, wordt “doe iets” zeker 1x doorlopen.

HO GENT

```
while (i < 10)
{
    instructie1;
    instructie2;
}
```

```
do
{
    instructie1;
    instructie2;
} while (i<10);
```