

Programmeren

Les 3

Terugblik vorige week

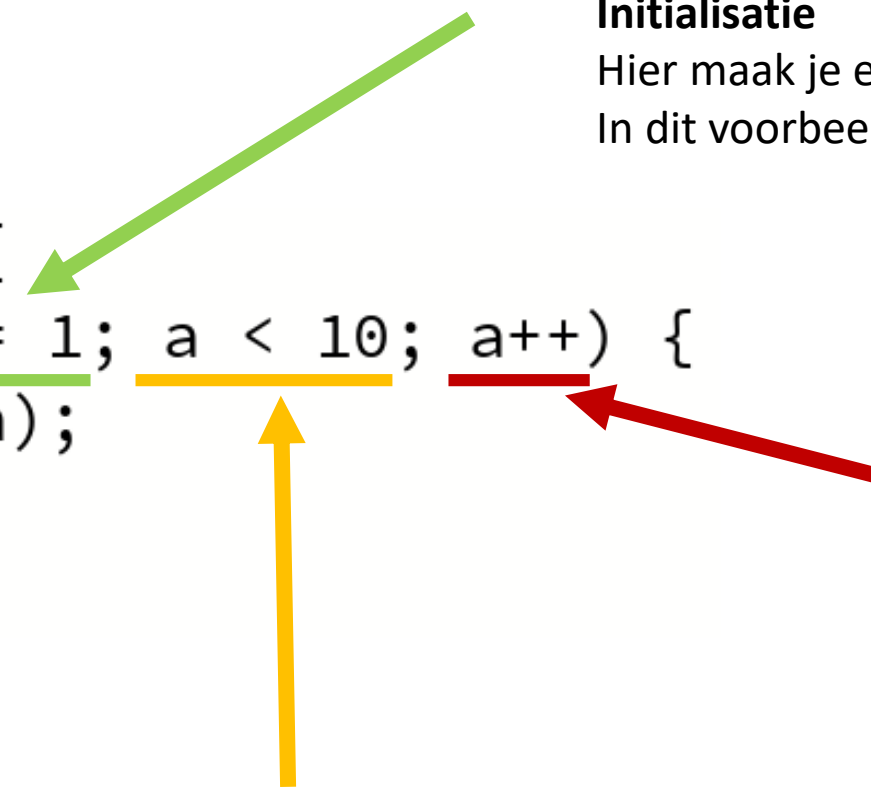
- Wat hebben jullie gemaakt?
 - Kun en wil je dat laten zien?

Loops

- Gebruik je om stukjes code meerdere keren uit te voeren. Meest gebruikte loops:
 - **For-loop**
Gebruik je meestal als iets een vooraf gedefinieerd aantal keren gedaan moet worden (dus je weet vooraf hoelang het moet)
 - **while-loop (vertaal while hier naar 'zolang ... waar is')**
Gebruik je meestal als je iets doet op basis van een waarde van een variable (bijvoorbeeld een boolean), maar je niet weet hoe vaak dat precies gaat gebeuren.

Loops - Voorbeeld

```
void setup() {  
  for (int a = 1; a < 10; a++) {  
    println(a);  
  }  
}
```



Initialisatie

Hier maak je een variabele aan, met een bepaalde waarde. In dit voorbeeld een variable met naam **a** en waarde **1**.

Increment

Dit doet hij elke stap, nadat de code in het blok is uitgevoerd

Conditie

Dit is de conditie die aangeeft hoelang de loop moet lopen (denk aan de if's van vorige week). In dit voorbeeld gaat hij door zolang **a** kleiner dan **10** is.

Loops – Voorbeelden

```
void setup() {  
    for (int a = 1; a < 10; a = a + 2) {  
        println(a);  
    }  
}
```

```
void setup() {  
    for (int a = 10; a > 0; a--) {  
        println(a);  
    }  
}
```

```
void setup() {  
    for (int a = 2; a <= 2048; a=a*2) {  
        println(a);  
    }  
}
```

Functies (ook wel methods genoemd)

Twee belangrijke redenen om functies te gebruiken:

- Je code wordt er leesbaarder van.
- Je kunt stukjes code hergebruiken, een soort 'subprogramma's'

```
void draw() {  
    tekenPenguin();  
}
```

```
void tekenPenguin() {  
    //Hier staat de code om de pinguin te tekenen  
}
```

Functies - argumenten

Functies kunnen argumenten (invoer) mee krijgen:

```
void draw() {  
    tekenPenguin(20, 20);  
    tekenPenguin(100, 100);  
}
```

```
void tekenPenguin(int x, int y) {  
    //Schrijf hier code dat op positie x,y een pinguin tekent  
}
```

Let op

De variabelen met de naam x en y zijn alleen binnen deze functie / methode beschikbaar!

Naam van de functie

Een functie kan iedere willekeurige naam krijgen. Gebruikelijk (in Java) is camelcasing te gebruiken. Dat betekent dat je het eerste woord met een kleine letter begint, alle volgende met een hoofdletter

Argumenten

Je kunt hier aangeven welke type variabelen je mee moet geven en welke naam ze geeft binnen de methode.

Functies – return waarden

Functies kunnen ook iets teruggeven:

```
void draw() {  
    tekenPenguin(optellen(10,10), 20);  
}
```

Aanroep van de functie

Hier roep ik de functie optellen aan. Die functie geeft een int terug en die int wordt op deze plek gebruikt (in dit geval dus **20**, omdat **optellen(10,10)** de waarde 20 terug geeft.

```
int optellen(int a, int b) {  
    return a + b;  
}
```

Argumenten

Deze ken je al. Een methode die iets teruggeeft kan ook argumenten hebben (maar hoeft niet).

Return-type

Het type dat de functie terug gaat geven. In dit geval een **int**
Als je hier **void** neerzet mag/kun je niets returnen.

Return

Een functie die een return value heeft moet aan het einde altijd iets returnen.

Functies – Voorbeeld

```
int leeftijd = 16;
if (isMeerderjarig(leeftijd)) {

}

boolean isMeerderjarig(int leeftijd) {
    if (leeftijd < 18) {
        return false;
    } else {
        return true;
    }
}
```

Functies – Separation of concerns

Verdeel de ‘concerns’ over je programma. Ofwel:

- Zorg dat elke methode verantwoordelijk is voor een eigen stukje programma.
- Deel het zo logisch mogelijk in!

Waarom?

Uitbreidbaarheid, leesbaarheid en aanpasbaarheid

Later gaan we hier nog meer verder door classes toe te voegen.

Funcities – Bestaande funcities

Misschien had je je inmiddels bedacht dat je al die tijd al funcities gebruikt hebt.

Zo zijn alle tekenfuncities (**line(..)**, **rect(..)**, etc) die je gebruikt hebt allemaal funcities.

Het enige is dat je die niet zelf geschreven hebt, maar dat de makers van processing die al voor je gemaakt hadden.

Random

```
void setup() {  
  float getal = random(10);  
  print(getal);  
}
```

```
void setup() {  
  float getal = random(5, 10);  
  print(getal);  
}
```

```
void setup() {  
  int getal = int(random(5, 10));  
  print(getal);  
}
```

Random aanroepen

Hier roep je de functie random aan (die Processing al voor ons gemaakt heeft).

Dit geeft een willekeurige float (decimaal getal) tussen 0 en de 10 (waarbij 10 zelf nooit voorkomt)

Lower en upper bound

Dit geeft een getal tussen de 5 (inclusief) en de 10 (exclusief)

Casting

De methode random geeft nu eenmaal een **float** terug. Als we een int willen hebben kunnen we de float casten naar een **int**.

Let op: Dit doet geen afronding, maar 'knipt' de decimalen er af.