

Inleiding in programmeren - p5.js

" The debate over whether the Net is good or bad for us fills the airwaves and the blogosphere. But for all the heat of claim and counter-claim, the argument is essentially beside the point: it's here; it's everywhere. The real question is, do we direct technology, or do we let ourselves be directed by it and those who have mastered it? "

Douglas Rushkoff
program or be programmed

1.Programmeren.

Denk aan alle boeiende dingen die je kan doen via een laptop, smartphone of tablet (muziek maken, games spelen, sociale media) en bedenk daarbij dat een belangrijke component van al die zaken programmeren is.

Nog niet zo erg lang geleden was programmeren weggelegd eerder voor ingenieurs, wetenschappers en *brainiacs* allerhande maar sinds een aantal jaren is er een **democratisering** gaande **met betrekking tot programmeren** en de toegang ertoe.

Vandaag de dag bestaan er specifieke **toepassingen om te leren programmeren** o.a. voor kinderen (scratch), ouderen en zelfs kunstenaars.

1.2 Wat is programmeren?

Programmeren valt goed te vergelijken met het **opstellen van een stappenplan bij het uitvoeren van een taak**. Soms is het ook makkelijk om te bedenken hoe je een kind iets zou aanleren. Stel je voor dat je een kind moet uitleggen hoe hij precies zijn tanden moeten poetsen. Binnen de context van programmeren is het kind de computer maar in essentie omschrijf je ook hier een serie te nemen stappen om tot een resultaat te komen.

In normale omstandigheden gebruiken we de computer vrij passief, we werken netjes binnen de kijtlijnen van een gekozen programma en gebruiken de machine als krachtige rekenmachine. **Leren programmeren geeft je de mogelijkheid om een gesprek aan te gaan met je computer**, in die zin dat je hem **zelf instructies kan opleggen** om iets uit te voeren.

Binnen de context van een al maar meer geautomatiseerde wereld wemelend van smart devices lijkt de mogelijkheid tot communiceren met je computer niet eens een overdreven luxe. (program or be programmed)

Hoe gaat zo'n gesprek in zijn werk?

1.2 Voltage.

Een van de meest fundamentele functies van een computer is de mogelijkheid om elektrische energie te controleren en te transformeren in een ander soort van energie zoals geluid, licht of beweging.

Op het meest elementaire niveau communiceren we met computers via twee numerieke waarden: een *0 (low voltage)* en een *1 (high voltage)*. Door verschillende sequenties op te bouwen rond die 0 en 1 verkrijgen we een continue flow van energie die resulteert in licht, geluid of een andere mogelijkheid eigen aan een computer.

Deze vorm van datarepresentatie kreeg de naam binaire of **Binary Code**.

Alhoewel de representatie van data in termen van low en high voltage zeer bruikbaar kan zijn heeft het als nadeel dat het moeilijk leesbaar en verstaanbaar is. Gelukkig stopt het representeren van data niet bij binaire code.

1.3 Abstracties.

Aangezien we weten dat een bepaalde sequentie van 0 en 1 iets representeren (bv: "00110101" = "53") kunnen we ook nieuwe sequenties maken zoals "01001000" en "01101001" die de letters "h" en "i" en dus samengesteld "hi" representeren. **Dit is precies wat software ontwikkelaars doen: een complexe set van data representeren in een (voor mensen) meer leesbare vorm.**

Naar dat proces wordt verwezen als het *maken van een abstractie*.

Een software programma is een set van instructies die beschikbaar gesteld worden voor een computer via source code. Source code is data die meestal erg lijkt op een soort tekstdocument opgemaakt aan de hand van een specifieke programmataal die het midden houdt tussen een taal die computers efficiënt kunnen verwerken en een taal die voor mensen makkelijk te begrijpen valt. **De taak van het creëren van source code is programmeren.**

Kunnen we dan geen abstractie maken waardoor we in natuurlijke taal kunnen communiceren met onze computer? Stel dat we een cirkel willen tekenen op een computer scherm, waarom kunnen we dat dan niet gewoon op die manier vragen:

"Computer, wil je een cirkel tekenen met een diameter van 55 pixels en op een locatie ongeveer aan de linker bovenkant van mijn scherm? bedankt."



1.4 Instructies.

Los van het feit dat een computer (mocht dat al mogelijk zijn) allicht moet lachen om woorden als "ongeveer" en "mijn" is dat laatste een niet zo evidente opdracht.

In functie van **snelheid, efficiëntie en duidelijkheid** communiceren we met onze computer op een andere manier. In het voorbeeld van de cirkel zouden (binnen p5 – wat we zullen gebruiken als leeromgeving) we een instructie ingeven:

```
ellipse(50, 50, 200, 200);
```

Die instructie verwijst naar een ander stuk code dat het tekenen van de cirkel effectief uitvoert in javascript. Die code zit een pak complexer in elkaar, verwijst naar een tekencontext en tekent effectief de cirkel via bezier curves:

```

4957  };
4958 p5.prototype.ellipse = function (x, y, w, h) {
4959   if (!this._doStroke && !this._doFill) {
4960     return;
4961   }
4962   w = Math.abs(w);
4963   h = Math.abs(h);
4964   var ctx = this.drawingContext;
4965   var vals = canvas.modeAdjust(x, y, w, h, this._ellipseMode);
4966   ctx.beginPath();
4967   if (w === h) {
4968     ctx.arc(vals.x + vals.w / 2, vals.y + vals.w / 2, vals.w / 2, 0, 2 * Math.PI, false);
4969   } else {
4970     var kappa = 0.5522848, ox = vals.w / 2 * kappa, oy = vals.h / 2 * kappa, xe = vals.x + vals.w,
4971     ye = vals.y + vals.h, xm = vals.x + vals.w / 2, ym = vals.y + vals.h / 2;
4972     ctx.moveTo(vals.x, ym);
4973     ctx.bezierCurveTo(vals.x, ym - oy, xm - ox, vals.y, xm, vals.y);
4974     ctx.bezierCurveTo(xm + ox, vals.y, xe, ym - oy, xe, ym);
4975     ctx.bezierCurveTo(xe, ym + oy, xm + ox, ye, xm, ye);
4976     ctx.bezierCurveTo(xm - ox, ye, vals.x, ym + oy, vals.x, ym);
4977     ctx.closePath();
4978   }
4979   if (this._doFill) {
4980     ctx.fill();
4981   }
4982   if (this._doStroke) {
4983     ctx.stroke();
4984   }
4985   return this;
4986 };
4986 p5.prototype.line = function (x1, y1, x2, y2) {
4987   if (!this._doStroke) {

```

Snelheid: Door data te abstraheren in een formaat dat meer leesbaar is voor mensen wordt diezelfde data minder leesbaar voor de computer of in het algemeen “de machine”. Om met data iets nuttig te kunnen doen moet de machine data converteren naar een leesbaar formaat. Al die conversies vragen uiteraard energie die meer efficiënt kan gebruikt worden voor andere berekeningen. Dit gegeven resulteert in het feit dat we snellere en meer krachtige computers zullen nodig hebben als we abstraheren op een niveau dat verder weg staat van machine code. Programmeren met behulp van natuurlijke taal (Natural Language Processing) zal dan ook bijzonder krachtige computers moeten incorporeren.

Efficiëntie: Taal is gebonden aan locatie. Iets wat voor ons een vrij normaal concept is als “friet special” behoeft in een ander land wellicht iets meer uitleg; een omschrijving. Stel je voor dat je die omschrijving gebruikt bij het bestellen in een frituur. “Een bord of pakje frieten met mayonaise, curry en-of tomatenketchup en fijngehakte ajointjes graag”. Daar zullen ze van opkijken. Vanuit efficiëntie oogpunt bestel je gewoon “friet special”. Computertalen zijn niet veel anders. Ze zijn het meest efficiënt als ze gebruikt worden onder specifieke voorwaarden en omstandigheden.

Duidelijkheid: Computers zijn zeer letterlijk. Ze accepteren instructies die leesbaar zijn eerder dan vage omschrijvingen van wat je hoopt te bekomen. De vraag: "Computer, wil je een cirkel tekenen met een diameter van 55 pixels ..." is in dat opzicht ook helemaal iets anders dan de erg duidelijke instructie `ellipse(50, 50, 200, 200)`

Hoe langer de instructie hoe meer mogelijkheden voor fouten en dubbelzinnigheden.

In natuurlijk taal bedienen we ons van woorden die gesteld binnen een bepaalde sequentie een zin vormen. Meer dan eens zijn er meerdere mogelijkheden om hetzelfde op een andere manier te zeggen.

Computertalen hebben niet die handigheid, ze zijn zeer strikt in het gebruik van syntax. Je kan in p5.js bv niet refereren naar

- `circle(50, 50, 200)` of
- `oval(50, 50, 200)` maar
- enkel naar `ellipse(50, 50, 200, 200)`

om een cirkel te tekenen.

1.4 Lower / higher level

Binnen programmeertalen zijn er algemeen gesproken **twee grote types: lower en higher level**. De lower level talen staan dicht bij **machine code** en zijn dus voor mensen moeilijker verstaanbaar. De higher level talen liggen dichter bij **natuurlijke taal**.

Lower level talen hebben doorgaans mogelijkheden op vlak van geheugenmanagement en de toegang tot compilers terwijl higher level talen gekenmerkt worden door een **eenvoudiger te lezen syntax**, doorgaans goede **documentatie** en een collectie van **bibliotheken** die de mogelijkheden met gebruik binnen dezelfde taal uitbreiden.

2. Tools en libraries.

2.1 Javascript.

JavaScript is een veelgebruikte scripttaal om enerzijds webpagina's interactief te maken en anderzijds web applicaties te ontwikkelen. De eerste versie dateert van 1995 en werd gemaakt door **Brendan Eich** specifiek voor gebruik binnen Netscape Navigator.

Er bestaat **client-side** en **server-side** JavaScript. Client side js wordt direct uitgevoerd via de browser van de bezoeker (client), server side js wordt eerst uitgevoerd op de server om nadien doorgestuurd te worden naar de browser van de bezoeker.

2.2 p5.js

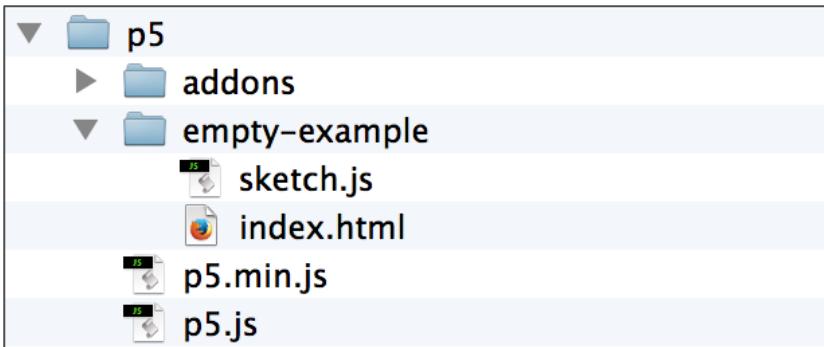
"p5.js is a Javascript library with the intention to make coding accessible for artists, designers, educators and beginners."

Een Javascript library of bibliotheek is een stuk source code meestal binnen een specifiek domein of met een specifieke functionaliteit. Er zijn dergelijke libraries rond oa audioprocessing, physics (box2D) en in dit geval specifiek ontworpen met als doel een tool te maken waarmee de basisbeginselen van programmeren kunnen geleerd worden.

p5 is in **nog steeds in ontwikkeling** en heeft een actieve community. Binnenkort komt er ook een code editor specifiek voor het programma maar daar wordt nog actief aan gewerkt. Uiteraard zijn er andere editors die we kunnen gebruiken.

P5.js kan je downloaden op p5js.org/download. Kies voor de **p5.js complete** link. Download en unzip de file.

De content van de unzipped folder zou er als volgt moeten uitzien:



- **addons** verwijst naar andere bibliotheken mbt audio en DOM.
- **empty-example** is een folder met daarin een HTML file en een JavaScript file die we als start kunnen gebruiken. We gebruiken de empty-example folder als werkfolder.
- Hernoem de folder naar WORKSTATION.
- **p5.js** is de library zelf die er voor zorgt dat alles behoorlijk werkt. Aanpassingen aan die file hoeft je niet te doen. De file bestaat in een origineel formaat en een formaat waaruit alle spaties gehaald zijn waardoor de file in zijn totaliteit kleiner is en dus minder tijd zal in beslag nemen om in te laden. (minified version: **p5.min.js**).

2.3 Brackets

Brackets is een open source tekst code editor die te downloaden valt op brackets.io

Een tekst code editor is een programma dat gebruikt wordt om effectief (syntax) code te schrijven. Open source wil zeggen dat de software en de broncode ervan vrij te gebruiken zijn.

1. Surf naar de bovenstaande url.
2. Klik op de link die refereert naar je persoonlijke operating systeem.
3. Download en installeer de software.
4. Open Brackets.
5. Onder File > Open Folder refereer je naar de p5 map.

Brackets zal de volledige folder openen en de files zichtbaar maken aan de linkerkant (donker). De files zelf zijn aanklikbaar en openen in het (witte) venster aan de rechterkant.

The screenshot shows the Brackets IDE interface. The title bar reads "empty-example/sketch.js — Brackets". On the left, the file tree shows "p5", "addons", "empty-example" (which contains "index.html" and "sketch.js"), "p5.js", and "p5.min.js". The main code editor window displays the following code:

```
1 function setup() {  
2     // put setup code here  
3 }  
4  
5 function draw() {  
6     // put drawing code here  
7 }  
8
```

Below the code editor, the status bar shows "Line 8, Column 1 — 8 Lines". To the right of the code editor, there are several icons: "live view -->" (highlighted in red), a green square icon, and other standard Brackets icons. The bottom right of the status bar shows "INS", "JavaScript", a checkmark icon, and "Spaces: 4".

We werken voor een webcontext, voor het bekijken van files hebben we dus een browser nodig. Op dit moment kunnen we alle files in code zien, een browser interpreteert die code en zet ze om naar webpagina.

index.html (in empty-example of WORKSTATION) is net zoals een elke HTML-pagina te bekijken in een browser.

Brackets heeft een **live view** die erop neer komt dat je simultaan kan kijken en naar je code in Brackets en naar de output ervan in een browservenster. De live view kan je aanzetten door op het bliksem icoon in de rechter boven hoek te klikken. Het programma opent dan een browser die de code zal weergeven.

De pagina openen zal resulteren in een lege pagina.

Nota: we zullen live view weinig gebruiken. **Simultaan gebruik van live view en de inspector tools van de browser is niet mogelijk.**

Bijkomend bij de installatie **installeren we ook volgende extensies:**

- **Emmet** (Sergey Chikuyonok)
- **Beautify** (Drew Hamlett)
- **JSHint** (Raymond Camden)

Extensies installeren kan je door op het lego icoon rechts bovenaan te klikken. Er zal een nieuwe tab geopend worden waar je de extensies kan zoeken via het zoek icoon.

2.4 Inspector tools

Gaan we terug naar `sketch.js` dan kunnen we iets uitvoeren om een bijkomende interessante tool binnen de browseromgeving in werking te zien.

Er staat wat default code in het document. Verwijder ze en sla het document op.

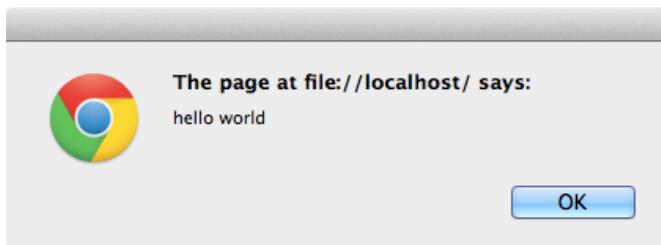
We gaan een standaard JavaScript functie oproepen die een tab zal openen met daarin een bericht.

1. Voeg volgende regel toe aan sketch.js:

```
alert("hello world");
```

2. **Sla de file op en refresh de browser.**

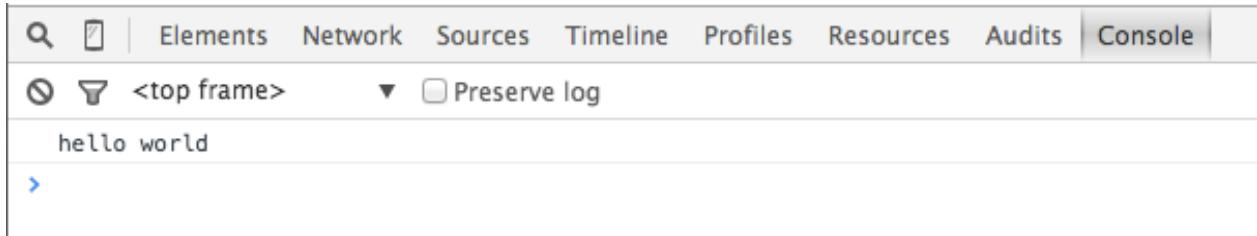
Normaal gezien zou je een berichtvenster moeten zien bovenop het browservenster met daarin de tekst *hello world* (en een referentie naar de pagina waarvan het bericht komt).



We kunnen eens andere functie uitproberen en de web inspector bekijken.

1. Verwijder de `alert` functie en vervang ze door
- ```
console.log("hello world");
```
2. Sla op en refresh het browservenster. Je krijgt een lege pagina.
  3. Open de web inspector tools via rechtermuisklik en de '**Inspect Element**' / '**Inspecteer element**' optie. De browser zal een nieuw venster openen met een aantal tabs specifiek bedoeld voor het opvragen van informatie over de pagina.

4. NOTA: Gesteld dat je de pagina in live view bekijkt zal je ook een melding krijgen dat de inspector en live view niet samen open kunnen staan. Live view zal zichzelf vervolgens afsluiten. Selecteer de optie 'Console' en bekijk het resultaat:



A screenshot of the Chrome DevTools interface, specifically the 'Console' tab. The tab bar at the top includes 'Elements', 'Network', 'Sources', 'Timeline', 'Profiles', 'Resources', 'Audits', and 'Console'. The 'Console' tab is active. Below the tab bar, there are two dropdown menus: one for frame selection set to '<top frame>' and another for log preservation set to 'Preserve log'. The main area shows the text 'hello world' on a single line.

## 2.5 (very) Quick html.

Doorgaans heeft een internetpagina een extensie '.html'. Die html is een afkorting voor '**HyperText Markup Language**', de **skeletstructuur van een standaard webpagina**.

De beschrijving van een HTML pagina gebeurt aan de hand van een aantal codes/tags waarvan we hieronder de belangrijkste binnen de context van p5 opsommen. De inspector laat toe om binnen het voorbeeld de andere tags op te zoeken.

- **<html></html>** is de container voor alle andere codes
- **<head></head>** omvat o.a de titel, referenties naar .js en .css files en meta info
- **<body></body>** binnen dit codepaar ligt de beschrijving van alle elementen van de pagina. Er zijn oa codes voor tekst (<p> </p>), titels <h1></h1>, beelden (<img>)
- **<canvas>** dit element wordt gebruikt om scripts in te visualiseren. Zoals eerder gesteld maakt p5 dit zelf. Uit de broncode: `document.createElement('canvas');`

## 3. p5.js library.

### 3.1 Basiselementen.

Laten we eerst stilstaan bij een aantal basiselementen van het programma.

#### 3.1.1 Setup en Draw.

Animatie over JavaScript in een webcontext verloopt op een specifieke manier die we later zullen bekijken. P5 werkt met het principe van 3 belangrijke functies:

- **setup()** { }
- **draw()** { }
- **preload()** { } // deze bekijken we later in de syllabus.

Je kan in setup() een aantal zaken als een canvas of variables initialiseren en de draw() functie gebruiken om alles uit te voeren waar repetitie in zit.

In het eerste voorbeeld wil ik een kleine rechthoek tekenen op positie 25,25 met een grootte van 50 op 50 pixels. Daar zit weinig visuele verandering in: de rechthoek zal zelfs binnen de repetitieve draw functie steeds op dezelfde plaats en grootte getekend worden. De structuur geeft ook andere mogelijkheden: we kunnen bv naast het tekenen van een figuur deze ook laten veranderen van dimensie, van kleur, enz.. Daarvoor hebben nood aan elementen die kunnen variëren, nl variabelen (waarover verder meer)

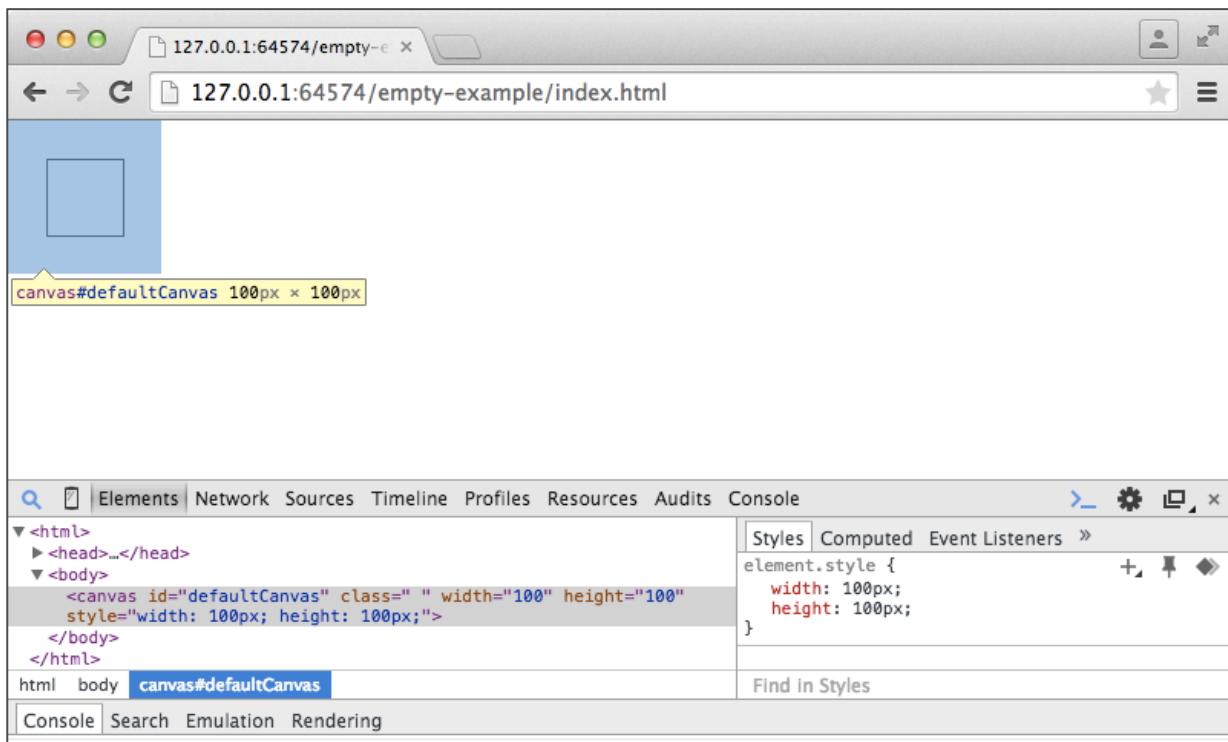
Code tussen setup() en draw() (en voor alle functies zoals we later zullen zien) zit altijd tussen accolades {}.

Voer volgende code in sketch.js, sla op en refresh de browser.

```
function setup() {
}

function draw() {
 rect(25, 25, 50, 50);
}
```

Normaal gesproken zou je een kleine rechthoek moeten zien in de linker bovenhoek van de webpagina. **Open de web inspector tools via rechtermuisklik** en de '**Inspect Element**' optie.



Klik op het zoekicoon of het vergrootglas (onderaan links) en richt de muispointer op de locatie van de kleine rechthoek. Merk op dat die oplicht en dat er in de Elements tab gerefereerd wordt naar een canvas tag met de id(aangegeven met het karakter '#') #defaultCanvas.

Zoals je kan merken gebeuren er een aantal zaken automatisch. In dit geval geven we zelf geen specifiek canvas op en dus maakt p5 er **automatisch** een van 100 bij 100 pixels. Uiteraard kunnen we alle standaard settings aanpassen aan onze noden.

**Het nulpunt van het venster ligt overigens in de linkerbovenhoek. Dat punt (0,0) verwijst naar een 2d locatie met 0 als waarde op de X-as en 0 als waarde op de Y-as.**

Dat coördinatensysteem is anders dan wat we kennen als papier voor technisch tekenen (millimeterpapier) waarbij het (0,0) in het midden van het blad ligt. We verwijzen naar het systeem gebruikt in p5 (en een hele resem andere programeer omgevingen) als het **Cartesiaans coördinatensysteem**.

### 3.1.2 Canvas.

Praktisch elke schets (dus ook een digitale) start bij een blad papier of een canvas. Binnen web is er een specifiek canvas object dat we kunnen gebruiken om via JavaScript rechtstreeks in dat venster te tekenen. In principe gebruiken we het canvas dus als tekenomgeving. We kunnen het canvas een vaste waarde geven of een relatieve waarde aangepast aan het apparaat waarmee je naar het canvas kijkt.

De functie om een canvas te maken is `createCanvas()` en wordt uitgevoerd in de setup van de sketch.

- `createCanvas(200,400);` // Een canvas van 200 bij 400 px:
- `createCanvas(displayWidth, displayHeight);` // Een canvas met relatieve waardes (display):
- `createCanvas(windowWidth, windowHeight);` // Een canvas met relatieve waardes (venster)
- `createCanvas(200,200,WEBGL);` // Een canvas in een webgl (3d) context:

### 3.1.3 Kleur.

Kleur in Processing kan je ingeven in verschillende (klassieke) formaten. In eerste instantie heb je

- **RGBa** (Red, Green, Blue,alpha), daarnaast
- **HSBa** (Hue, Saturation, Brightness, alpha) en tenslotte een
- **hexadecimale** waarde.

De laatste variant gaan we niet vaak gebruiken. Bijkomend kan je optioneel (voor RGB en HSB) een alpha waarde meegeven wat refereert naar de transparantie van de kleur. **RGB is de standaard setting**. Die is anders te definiëren door een `colorMode()` functie op te nemen.

```
colorMode(HSB, 255); // hsb color value binnen de schaal 0 – 255.
```

Kleur is bruikbaar binnen een aantal functies. Je kan er een vorm of stuk tekst een invulling mee geven of je kan het gebruiken om een achtergrond kleur te geven:

Een color functie accepteert:

- **1 argument**(voor grijswaarden)
- **3 argumenten**(RGB, HSB) of
- **4 argumenten** (RGB, HSB met A voor transparantie)

Stel dat we een oranje achtergrond willen kunnen we de functie invullen met volgende waardes:

```
colorMode(RGB);
background(255,130,0);
```

### **3.1.3a: Functies die kleur behandelen:**

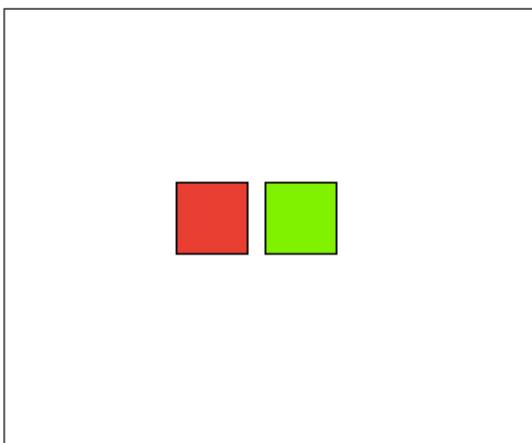
Binnen volgende functies kan je kleur gebruiken:

- fill()
- stroke()
- background()

#### **Wees voorzichtig:**

Als je een kleurfunctie als fill(255,0,0) ergens opneemt in je programma wil dat zeggen dat het programma ALLES wat een kleur kan krijgen in het rood zal zetten.

Om dat te vermijden, je wil bijvoorbeeld een groen naast een rood vierkant zal je de fill functie **twee maal moeten uitvoeren**. Een keer voor het tekenen van de ene rechthoek en een keer voor het tekenen van de andere rechthoek.



```
stroke(0);
fill(255,0,0);
rect(100,100,40,40);
fill(0,255,0);
rect(150,100,40,40);
```

Stroke en fill hebben ook **gerelateerde functies**.

Zo heeft de **lijnuitvoering** een aantal mogelijkheden op vlak van **lijndikte**, caps en join. Ten slotte kan je ook aangeven dat lijnen niet getekend moeten worden.

- `strokeWeight(5.0);` // lijndikte.
- `strokeCap(ROUND);` // of SQUARE or PROJECT
- `strokeJoin(ROUND);` // of MITER or BEVEL
- `noStroke();` // contouren, lijnen worden niet getekend.
- `noFill();` //vormen worden niet ingekleurd.

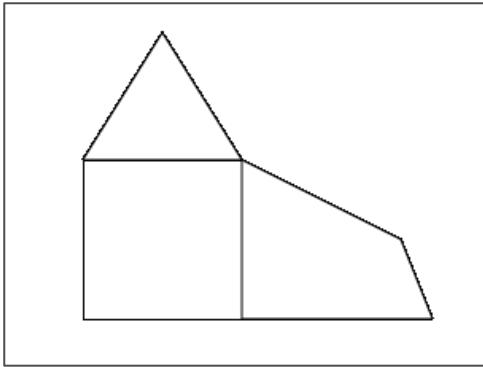
### 3.1.4 Primitives

p5.js heeft een aantal instructies voor het maken van verschillende vormen. Er bestaan basisvormen ( basic primitives ) voor 2D (en 3D).

- `point(x, y);` tekent een coördinaat/punt in de ruimte. Het eerste argument is de horizontale positie, het tweede de verticale.
- `line(x1, y1, x2, y2);` tekent een lijnverbinding tussen twee coördinaten/punten in de ruimte.
- `ellipse(x, y, width, height);` tekent een cirkel op een specifieke locatie met een specifieke breedte en hoogte.
- `rect(x, y, width, height, rounding);` tekent een rechthoek op een specifieke locatie met een specifieke breedte en hoogte. Optionele parameters gaan naar afronding van de hoeken.
- `quad(x1, y1, x2, y2, x3, y3, x4, y4);` tekent een polygoon met vier zijden die een verbinding zijn tussen vier punten.
- `triangle(x1, y1, x2, y2, x3, y3);` tekent een driehoek tussen drie punten.
- `arc(x, y, width, height, start, stop, mode);` tekent een curve tussen twee punten. Er zijn een aantal tekenmodi: OPEN, CHORD en PIE

Stel dat we een eenvoudig gebouw willen tekenen dat er als volgt uitziet:

- een rechthoek voor de linkerkant van de constructie.
- een driehoek voor het dak.
- een quad voor de rechter en meer grillige kant.



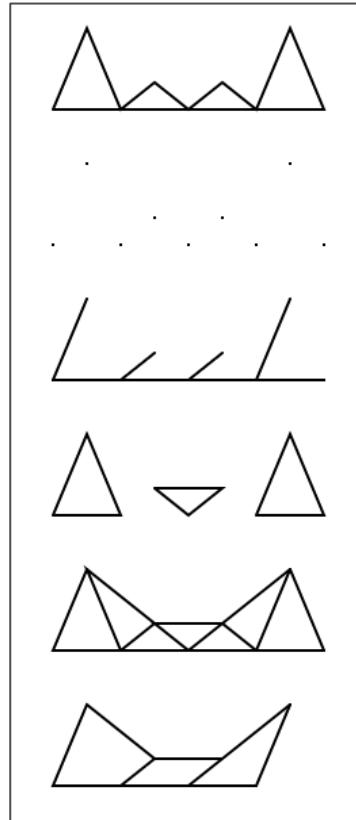
```
rect(50,100,100,100);
triangle(50,100,100,20,150,100);
quad(150,100,250,150,270,200,200,150,200);
```

### 3.1.5 Complexe vormen.

Meer complexe vormen kunnen gemaakt worden via `beginShape()` en `endShape()` functie. De **basis is een reeks van punten** / vertexes (x,y) die **op een specifieke manier verbonden** worden. De manier / vorm hoe die verbinding moet gebeuren wordt als argument meegestuurd met `beginShape()`.

Alle vormen hiernaast zijn bekomen via deze procedure:

```
beginShape();
vertex(100, 290);
vertex(125, 230);
vertex(150, 290);
vertex(175, 270);
vertex(200, 290);
vertex(225, 270);
vertex(250, 290);
vertex(275, 230);
vertex(300, 290);
endShape(CLOSE);
```



`beginShape();`

`beginShape(POINTS);`

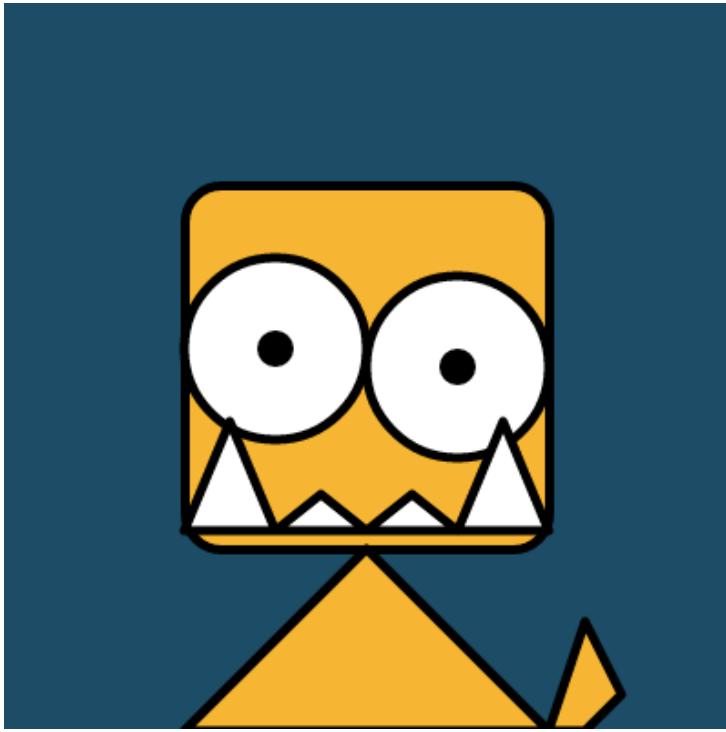
`beginShape(LINES);`

`beginShape(TRIANGLES);`

`beginShape(TRIANGLE_ST  
RIP);`

`beginShape(QUADS);`

### 3.1.6 Combinatie:



```
function draw() {
 background(0, 20, 120);

 stroke(0);
 strokeWeight(5.0);
 fill(255, 182, 0);
 rect(200, 200, 200, 200, 20);
 fill(255);
 ellipse(150, 190, 100, 100);
 ellipse(250, 200, 100, 100);
 fill(0);
 ellipse(150, 190, 15, 15);
 ellipse(250, 200, 15, 15);
 fill(255, 130, 0);
 triangle(100, 400, 200, 300, 300, 400);
 quad(300, 400, 320, 340, 340, 380, 320, 400);

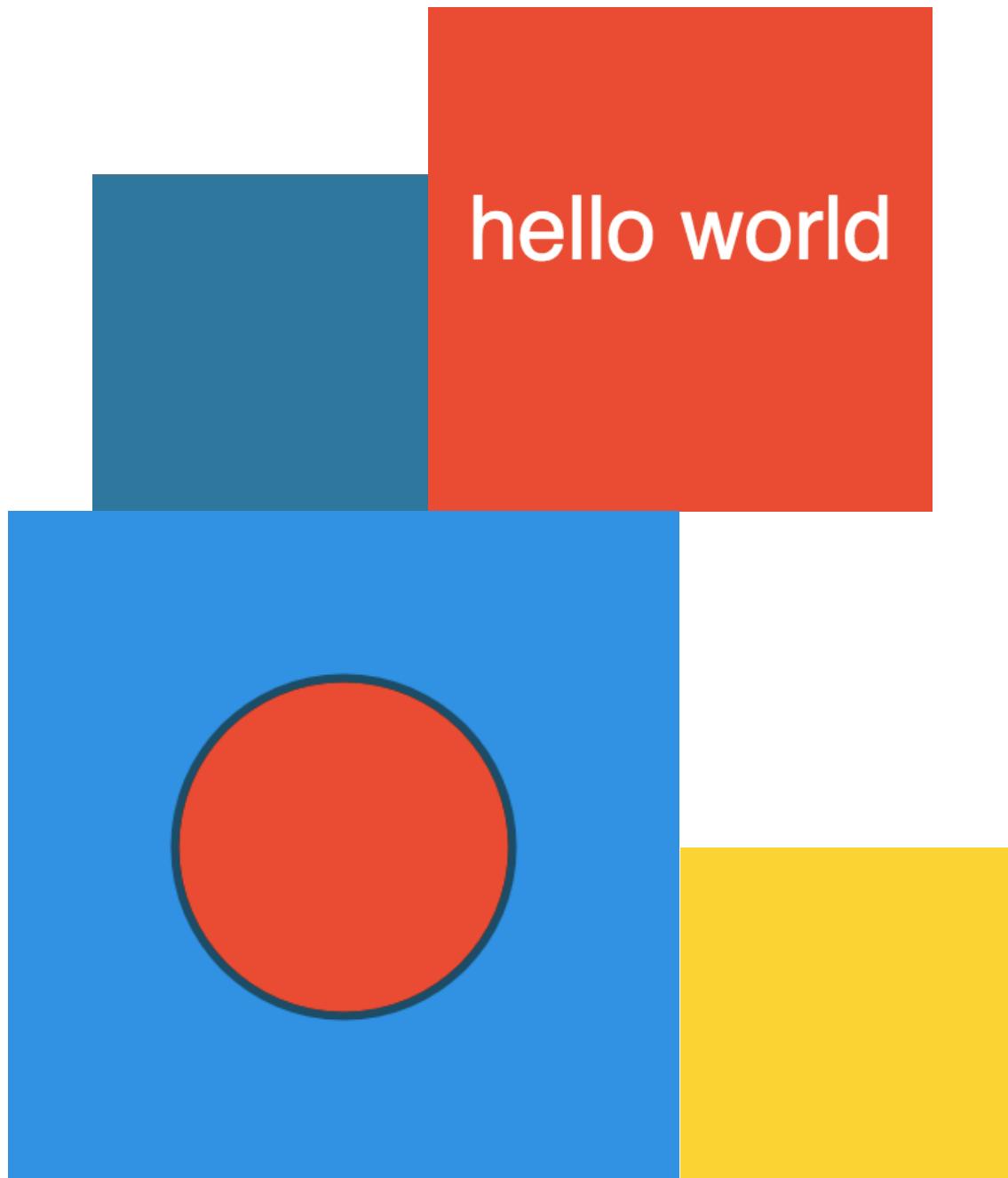
 fill(255);
 mond();
}
```

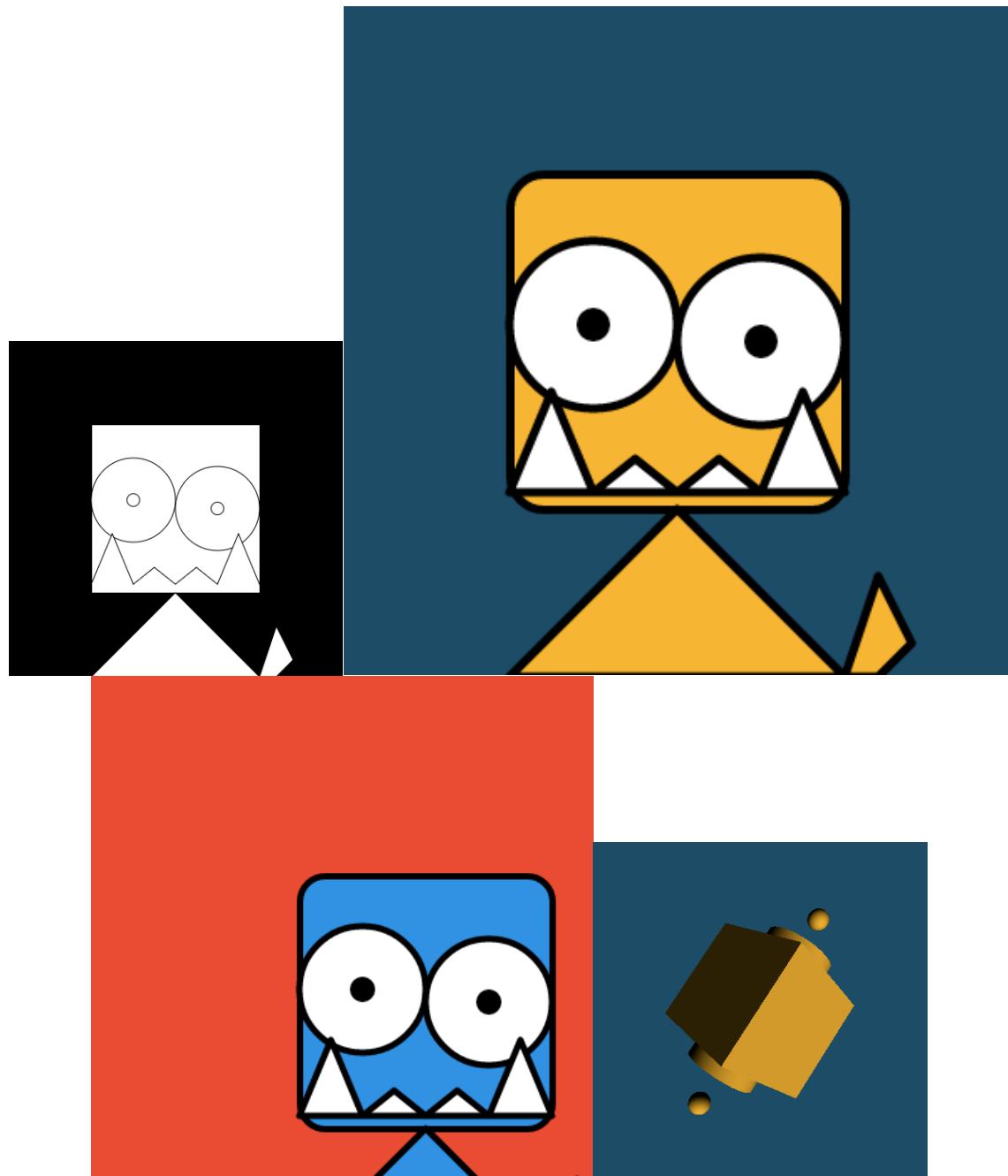
```
function setup() {
 createCanvas(400, 400);
 rectMode(CENTER);
 ellipseMode(CENTER);
}
```

```
function mond(){
 beginShape();
 vertex(100, 290);
 vertex(125, 230);
 vertex(150, 290);
 vertex(175, 270);
 vertex(200, 290);
 vertex(225, 270);
 vertex(250, 290);
 vertex(275, 230);
 vertex(300, 290);
 endShape(CLOSE);
}
```

## 3.2 oefeningen

1. 01abc\_basic\_hello
2. 02ab\_basic\_color
3. 03abcd\_basic\_primitives





## 4. Variabelen

*A variable is a lovely, yellow post-it note, on which is written the message: I am a variable. Write your information on me.*

*Daniel Shiffman*

Een computer heeft geheugen. Dat noemen we zo omdat het dat is wat een computer gebruikt om dingen te onthouden. **Technisch gesproken is een variabele een (named) pointer naar een locatie in de computer (memory adress) waar data wordt opgeslagen.** Aangezien computers maar 1 instructie per keer kunnen uitvoeren laat een variabele toe om informatie op te slaan op een bepaald moment in het programma om het dan vervolgens verder in het programma opnieuw te kunnen oproepen en gebruiken. De kracht van een variabele is niet enkel dat ze iets kan opslaan, maar ook en dat ze op een bepaald moment een nieuwe waarde kan krijgen, dat ze kan variëren.

Zo kan je de dimensie van een vierkant opslaan als een variabele om ze bijvoorbeeld elke seconde te laten veranderen naar een andere dimensie.

Variabelen kunnen primitieve waarden stockeren maar ook referenties naar objecten of arrays. We kijken voorlopig alleen naar de primitieve waardes.

### 4.1 Declaratie en Waarde toekenning.

Het gebruik van variabelen kan je eenvoudig samenvatten in twee stappen:

1. Declaratie of declaration
2. Waarde toekenning of Assignment

Variabelen worden **gedeclareerd door eerst te verwijzen naar het type 'var' gevuld door de naam van de variabele.**

De naam van een variabele is 1 woord zonder spaties en mag niet starten met een cijfer. Ze kunnen ook geen interpunctie of speciale karakters (uitgezonderd het underscore teken) in de naam meekrijgen.

Een variabele kan ook een waarde krijgen op basis van een andere variabele ( $x$  is gelijk aan  $y$ ), of door een wiskundige bewerking uit te voeren met meerdere variabelen( $x$  is gelijk aan  $y$  plus  $z$ ).

## 4.2 Voorbeelden.

```
var mijnVariabele; // Declaratie
mijnVariabele = 5; // Waarde toekenning
```

Een aantal andere variabelen:

- `var count = 0;` // Declareert een nummer genaamd count, met als waarde 0
- `var letter = 'a';` //Declareert een char genaamd letter met als waarde a
- `var happy = false;` // Declareert een boolean genaamd happy, met als waarde false
- `var txt = "hello world";` //Declareert een string genaamd txt met als waarde de tekst hello world.
- `var getal = 4.3;` // Declareert een nummer genaamd x, met als waarde 4.3
- `var y;` // Declareert een nummer genaamd y en doet geen waardebepaling
- `var x = y+5.2;` // Kent de waarde x plus 5.2 toe aan y (die daarvoor gedeclareerd werd)
- `var z = x* y +15.0;` // Declareert een float genaamd z, de waarde die ze mee krijgt is x vermenigvuldigd met y plus 15.0

## 4.3 Systeem variabelen.

p5 heeft ook een aantal handige **systeemvariabelen** die instaan voor data die bruikbaar zijn binnen veel sketches (bv de dimensies van het display venster, de key op het keyboard die ingedrukt werd. Als je met additionele variabelen werkt is het aangewezen om volgende namen dan ook NIET te gebruiken.

- `width`: verwijst naar de breedte (in pixels) van het display venster.
- `height`: verwijst naar de hoogte (in pixels) van het display venster.
- `frameCount`: verwijst naar het aantal frames die al voorbij zijn.
- `frameRate`: verwijst naar de aantal frames / seconde.
- `displayWidth`: verwijst naar de breedte (in pixels) van het computerscherm.
- `displayHeight`: verwijst naar de hoogte (in pixels) van het computerscherm.

- `mouseX` en `mouseY` zijn nummer waarden die de coördinaten van de computermuis weergeven.
- `pmouseX` en `pmouseY` zijn floating waardes die de vorige coördinaten van de computermuis weergeven.
- `mousePressed`: is een boolean (true/false) die aangeeft of je met de muis klikt.
- `mouseButton`: verwijst naar welke muisknop ingedrukt is: left, richt, center.
- `key`: verwijst naar de toets die je indrukt op het keyboard.
- `keyCode`: verwijst naar de numerieke code van diezelfde toets.
- `keyPressed`: is een boolean (true/false) die aangeeft of er een toets ingedrukt staat.

## 4.4 Wiskunde operatoren.

p5 laat het gebruik van een aantal **wiskunde operatoren** toe. Deze laten toe om verschillende getallen op te tellen, te delen enz. De meest gebruikte functies zijn:

- **=** : is gelijk aan
- **+** : optellen : bv:  $z = x + y;$
- **-** : aftrekken : bv:  $z = y - x;$
- **/** : delen : bv:  $z = x / 5.0;$
- **\*** : vermenigvuldigen : bv:  $z = y * 2.0;$
- **++** : verhoging met 1 : bv:  $x++;$
- **--** : verlaging met 1 : bv:  $x--;$
- **+=** : verhoging met een factor niet gelijk aan 1 : bv:  $x+=5;$
- **-=** : verlaging met een factor niet gelijk aan 1 : bv:  $x-=y;$
- **%** : modulo of restdeling. bv:  $10\%5 = 0$ ; (want 10 is deelbaar door 5 met een rest van 0);

Modulo of de restwaarde bij een deling wordt gebruikt om bijvoorbeeld iets binnen de limieten van het scherm te houden of om iets uit te voeren met een bepaald interval.

```
var a = 11;
var b = 5;
var c = a%5;
```

// resulteert in 1 want 11 gedeeld door 5 is 2 met een restwaarde van 1.

## 4.5 Voorbeeld

Stel dat we een balletje willen laten bewegen op het scherm. Wat hebben we dan nodig:

- een canvas dat steeds opnieuw getekend wordt.
- een bolletje ==> ellipse functie die een locatie meekrijgt (x,y) en een dimensie (w,h)
- een locatie ==> x,y positie van het bolletje: 2 variabelen --> x,y
- een snelheid ==> duwt het bolletje vooruit: 1 variabele --> s
- een setup() en draw() structuur omdat we iets niet 1 keer willen uitvoeren maar het balletje willen laten bewegen.

in code:

```
var x,y; // initialiseert de variabelen x en y

function setup() {
 createCanvas(400, 400);
 x = 200; // geef de variabele een startwaarde
 y = 200;
}

function draw() {
 background(255);
 x += 2; //verhoog de waarde telkens met 2
 ellipse(x, y, 8, 8); // gebruik de variabele om de circle te tekenen.
}
```

Wat doet p5 nu precies? Bij opstart van het programma worden 2 variabelen gemaakt en in een eerste frame (setup functie) krijgen ze een startwaarde mee. Daarnaast maken we een canvas van 400 bij 400 pixels.

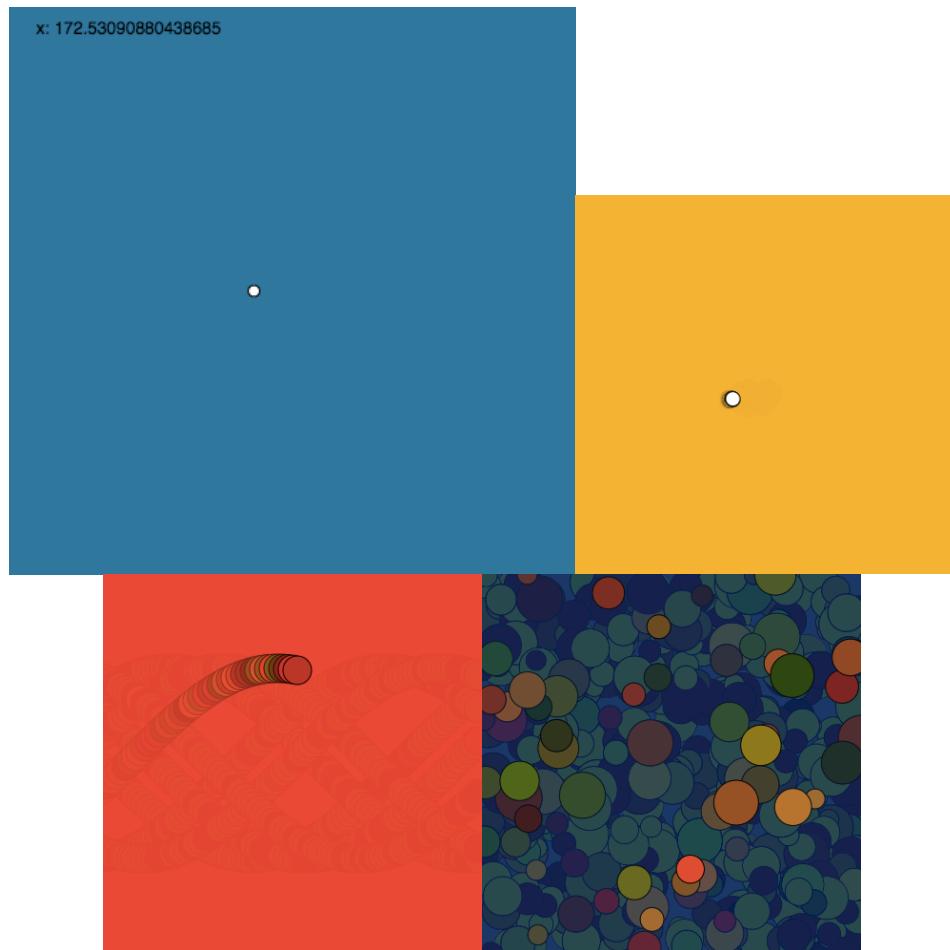
Vanaf dan krijgen we een zichzelf herhalende draw functie die de waarde van de initiële x positie aanpast door de oude waarde op te tellen met 2. Op frame 1 is x 50, op frame 2 is x al 52, frame 3 is x 54 enz.

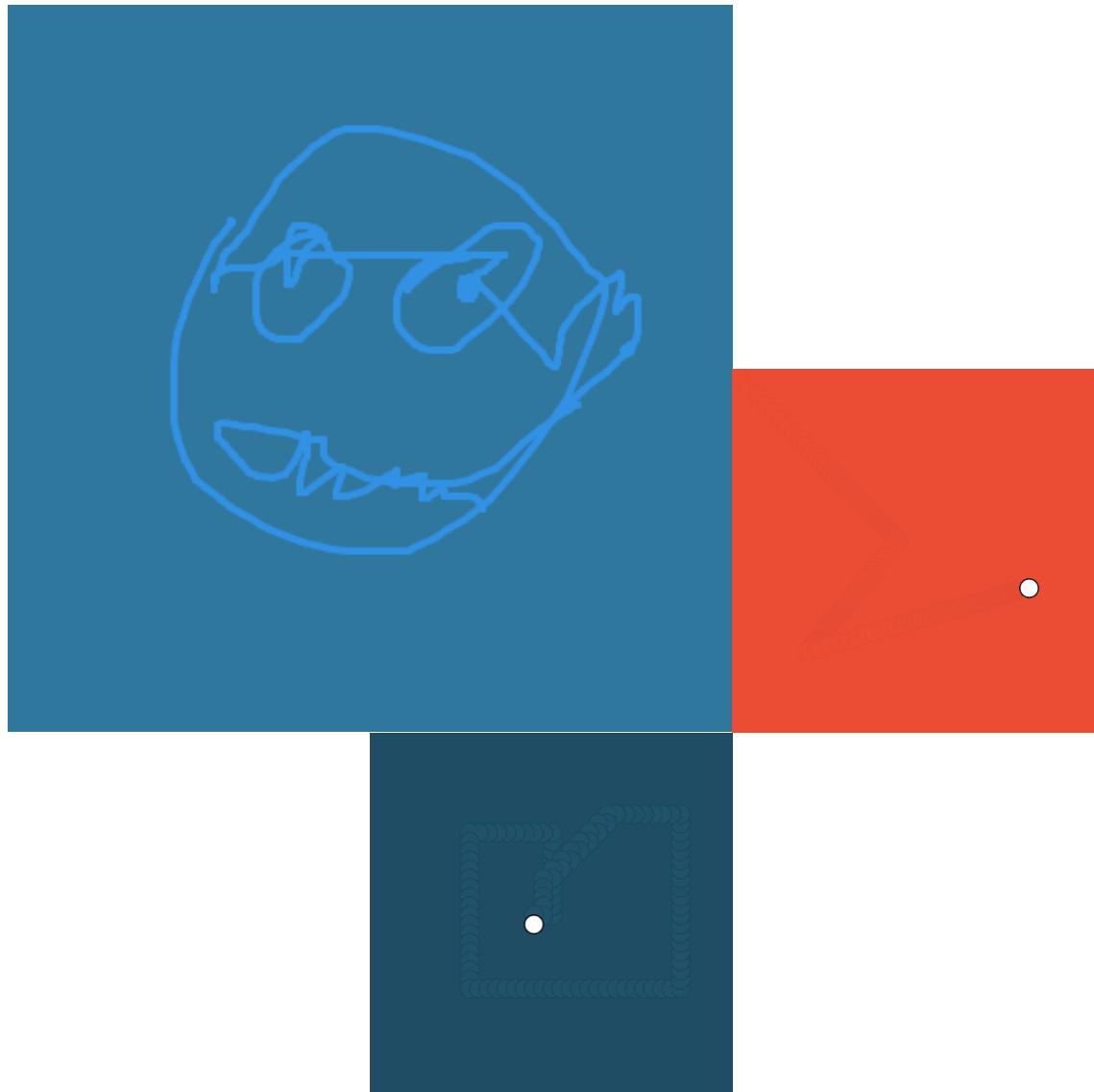
Door diezelfde variabele te gebruiken om het bolletje op die locatie te tekenen krijgen we het visuele resultaat dat het bolletje naar rechts beweegt.

De background functie zorgt ervoor dat we steeds opnieuw naar een nieuw canvas kijken en er dus tegelijkertijd maar 1 bolletje getoond wordt.

## 4.7 Oefeningen

1. 04abc\_basic\_variables
2. 04de\_basic\_system\_variables
3. 04f\_basic\_keys





## 5. Condities.

### 5.1 Branching.

Branching of **conditioneel programmeren** is het proces van code schrijven dat de mogelijkheid heeft om een bepaald traject te volgen gebaseerd op een set van condities opgenomen in het programma.

Die condities kunnen ook gestuurd worden door condities buiten het programma. Bijvoorbeeld via interactie met de gebruiker. In gaming krijg je vaak dat principe toegepast, het spel evolueert op basis van de beslissingen van de speler.

De meest voorkomende is de if statement die er in syntax als volgt uitziet:

```
if(expression){
 // doe iets
}
```

Uit expression (die we de **conditie** noemen) komt een boolean true / false op basis van waarop een actie wel of niet wordt uitgevoerd. Een if statement kan aangevuld worden met een else statement die aangeeft wat (bij false) wel moet worden uitgevoerd:

```
if(expression){
 // doe iets
}else{
 // doe iets anders
}
```

Tenslotte kan je de if statement nog uitbreiden met een else if statement:

```
if(conditie){
 // doe iets
}else if(conditie 2){
 // doe iets anders
}else{
 // doe nog iets anders
}
```

**Een eenvoudig if - else voorbeeld** waarbij twee verschillende zinnen getoond worden op basis van de locatie van de muis ziet er als volgt uit:

```
function setup() {
 createCanvas(300,300);
}
```

```
function draw() {
 background(255);
 line(0,height/2,width,height/2);
 if(mouseY >= height/2) {
 text("muis is in de onderste helft van het display venster");
 }else{
 text("muis is in de bovenste helft van het display venster");
 }
}
```

## 5.2 Expressies en operators.

Bij het schrijven van condities kunnen we **een vergelijking implementeren of meer dan 1 expressie opnemen in de conditie**.

Een vergelijking kan je uitvoeren via een **comparison operator**

- **<** : kleiner dan
- **>** : groter dan
- **<=** : kleiner of gelijk aan
- **>=** : grote of gelijk aan
- **==** : gelijk aan
- **!=** : niet gelijk aan

Voor het opnemen van meerdere expressies in 1 conditie maken we gebruik van **logical operators**:

- **&&** : logical AND
- **||** : logical OR
- **!** : logical NOT

Deze operators kunnen gebruikt worden binnen een conditie om opnieuw een waarde true of false terug te geven maar dan gebaseerd op de context in dewelke ze gebruikt worden.

Een logical **AND** zal enkel true teruggeven als beide expressies als true geëvalueerd kunnen worden.

- `(5 > 3 && 3 > 1)` //Geeft true terug aangezien beide condities waar zijn.
- `(5 > 3 && 3 < 1)` //Geeft false terug aangezien 1 van de condities niet waar is.

Een logical **OR** zal true teruggeven als op zijn minst 1 van de condities waar is.

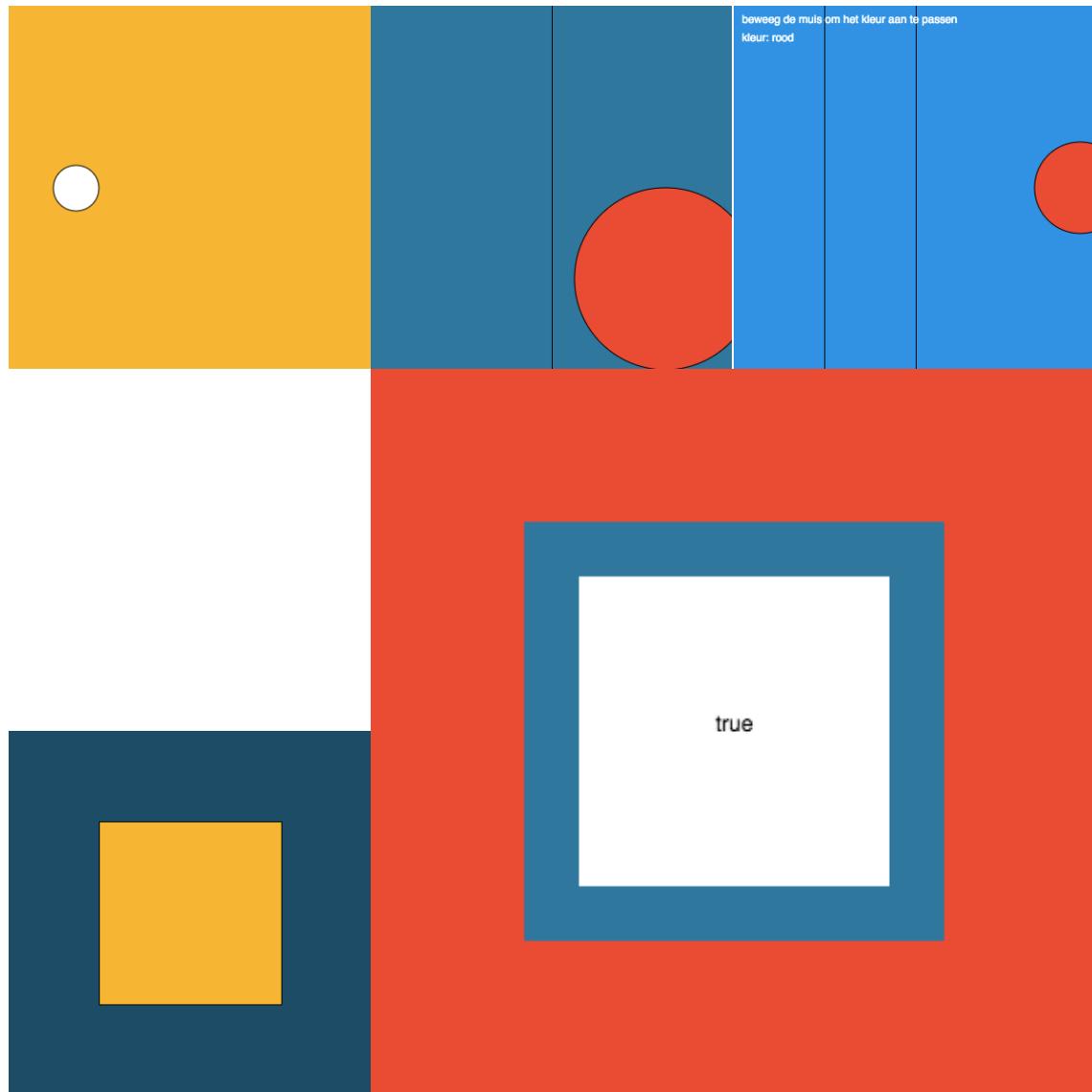
- `(5 > 3 || 3 > 1)` //Geeft true terug aangezien beide condities waar zijn
- `(5 > 3 || 3 < 1)` //Geeft true terug aangezien de conditie links waar is.
- `(5 < 3 || 3 < 1)` //Geeft false terug aangezien beide condities fout zijn.

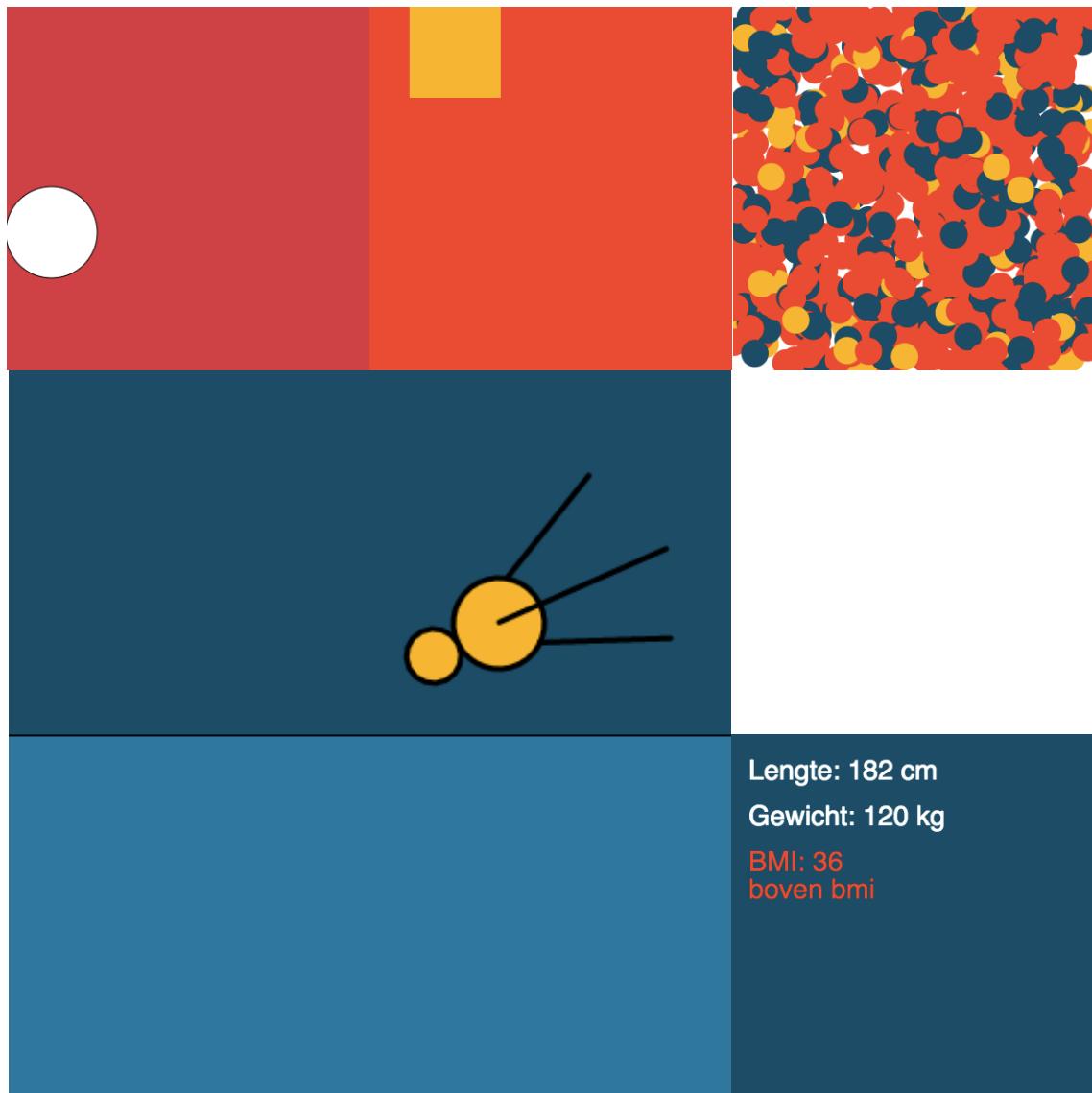
Een logical **NOT** werkt op een iets andere manier. Het zal de omgekeerde waarde teruggeven van de expressie die ze evalueert.

- Als `(x>1)` als true evalueert dan zal `!(x>1)` als false geëvalueerd worden.

## 5.3 Oefeningen

1. 05abc\_conditional\_branching (comparion)
2. 05de\_conditional\_branching (logical)
3. 05fghij\_conditional\_branching (combinaties)





## 6. Functies.

Hoe meer code we schrijven hoe groter de draw functie zal worden. In een aantal bovenstaande voorbeelden zou sommige functionaliteit even goed in nieuwe functies geschreven kunnen worden. Het gebruik van functies maakt code overzichtelijk en leesbaar.

### 6.1 Niets nieuws onder de zon.

We gebruiken al de hele tijd functies. Als we rect(100,200,50,50) als statement ingeven roepen we in feite de functie rect() aan, een built-in functie van de library.

De mogelijkheid om een rechthoek te tekenen bestaat namelijk niet zomaar maar is geïmplementeerd in het programma. Een van de krachtige eigenschappen van p5 is de bibliotheek van beschikbare functies waarvan we er al een aantal hebben gezien. Naast built-in functies zijn er ook user defined functies.

Een functie definitie of declaratie bestaat uit drie fundamentele delen:

1. return type (doorgaans function)
2. functie naam
3. argumenten

in pseudocode ziet een functie er als volgt uit:

```
returnType naam_van_de_functie (argumenten) {
 // code van de functie.
}
```

Stel dat we een functie willen maken met een eenvoudige taak zoals het teken van een rode cirkel:

```
function drawARedEllipse() {
 fill(255,0,0);
 ellipse(width/2,height/2,50,50);
}
```

**Let op: het definiëren van een functie voert ze nog niet uit.** We zullen de functie ook moeten aanroepen binnen het programma. Meestal zal dat in de draw() functie gebeuren aangezien die functie *continue is (loop)*.

vb:

```
//p5 setup functie

function setup() {
 createCanvas(400,400);
}

// definiëren van de eigen functie

function drawARedEllipse() {
 fill(255,0,0);
 ellipse(width/2,height/2,50,50);
}

//p5 draw functie

function draw() {
 background(0);
 // aanroepen van de functie
 drawARedEllipse();
}
```

## 6.2 Argumenten.

Als we de waarde van locatie van de cirkels dynamisch willen maken kunnen we die waarde meesturen als **argument**.

Een argument is de declaratie van een variabele binnen de haakjes van de functie definitie. De tijdelijke variabelen dienen enkel binnen de functie en kunnen daarom niet opgevraagd worden buiten de functie.

Verander de nieuwe functie met volgende code:

```
function drawARedEllipse(x, y) {
 fill(255, 0, 0);
 ellipse(x, y, 50, 50);
}
```

Uiteraard dient de functie met argumenten op een andere manier aangeroepen te worden. De argumenten van de functie verwachten namelijk een invulling. Stel dat we de cirkel de muis coördinaten willen laten volgen kunnen we voor de tijdelijke variabelen x en y de mouseX en mouseY waarde

meegeven:

```
drawARedEllipse(100, 100);
```

of

```
drawARedEllipse(mouseX, mouseY);
```

belangrijk om te onthouden bij het doorgeven van parameters via argumenten:

- Er moeten altijd evenveel parameters worden doorgegeven als er argumenten zijn.
- De waarde die je doorgeeft als parameter kan een waarde op zich zijn (30, 45, 5.3, "a", enz) maar kan ook een variabele zijn (x, speed, mouseX, enz.) en zelfs het resultaat van een functie (4 \* y/5.0, random(-5,5) of een functie)
- Argumenten zijn lokale variabelen dus enkel toegankelijk en opvraagbaar binnen de functie.

## 7. Iteratie.

We hebben eerder al gezien dat de draw functie in essentie een looping procedure is. Bijkomend kunnen we subroutines als loop of iteratie implementeren. Stel dat je een reeks van vierkanten wil tekenen die horizontaal naast elkaar staan dan zou je kunnen coderen als volgt:

```
rect(20, 200, 20, 20);
rect(40, 200, 20, 20);
rect(60, 200, 20, 20);
```

Voor een klein aantal kan dat nog aanvaardbaar lijken, maar wat doe je als je 2000 vierkanten wil tekenen.

### 7.1 For loop

Programmeren heeft twee belangrijke principes voor iteratie of loops: de **for() structuur en de while() structuur**. De while() structuur kan je zien als een vereenvoudigde versie van een for() structuur. We focussen ons op de for() structuur. Een oefening op while() kan je terugvinden bij de voorbeeldfiles.

De for() structuur wordt gebruikt om een waarde te itereren, dat wil zeggen, de waarde te laten veranderen volgens een herhalend patroon. De for() structuur wordt ook de for() loop genoemd en volgt doorgaans het volgende protocol in pseudocode:

```
for(init; test; update){
...statements
}
```

De for() functie krijgt drie argumenten mee die onderscheiden zijn door een punt komma (om aan te tonen dat het om drie verschillende argumenten gaat).

- **init** wordt gebruikt om initialisatie te beschrijven. In de for() loop wordt var beschreven door een tijdelijke variabele te initialiseren en daaraan een waarde toe te kennen. Die variabele kan dan gebruikt worden in de volgende argumenten.
- **test** wordt gebruikt om een conditie te beschrijven. Die conditie (zoals elke

conditie) zal true of false evalueren. Als de conditie true evalueert zal de for() structuur uitgevoerd worden, als ze false evalueert zal Processing verder gaan met de statements die de for() structuur volgen.

- **update** beschrijft hoe de tijdelijke variatie moet worden aangepast per iteratie. Meestal is dat een waarde verhoging met

voorbeeld:

```
for(var i = 0; i < 10; i++){
 console.log(i);
}
```

zal 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 printen.

Het gebruik van een for() loop als een counter (zoals hierboven) is een interessante manier om waardes te updaten binnen een sketch maar for() loops kunnen wel meer dan een eenvoudige counter maken.

Onderstaand voorbeeld gebruikt een for() loop om een onregelmatig vorm te beschrijven:

```
beginShape();
for(var i = 0; i < 10; i++){
 vertex(random(width), random(height));
}
endShape(CLOSE);
```

## 7.2 Transformaties.

Binnen for() loops (maar ook daarbuiten) wordt vaak gebruik gemaakt van transformatie functies. De gebruikelijke transformaties bestaan als aparte functie:

- **translate(x,y)** voert een verplaatsing op de x y as uit. Er is mogelijkheid tot uitbreiding naar een derde coördinaat in WebGL.
- **rotate(a)** voert een rotatie uit op basis van degrees() of radians()
- **scale(x, y)** voert een schaling uit op de x en y dimensie.

Door transform statements te gebruiken krijgen we als gevolg dat het volledige coördinaten systeem getransformeerd wordt. In een aantal gevallen willen we dat vermijden. Twee functies in Processing die dat probleem oplossen zijn de functies **push()** en **pop()**.

De push() functie zal, als ze opgeroepen wordt vooraleer een transformatie gebeurd, de huidige waarde van het coördinaten systeem opslaan. Daarna ben je vrij om elke transformatie uit te voeren die nodig is (translatie, rotatie, scaling) en de component te renderen (bv een rechthoek te tekenen). Vervolgens sluit je de procedure af door een pop() functie op te roepen die het coördinaten systeem terugzet naar de waarden die ze had alvorens de laatste push() opgeroepen werd.

Het gebruik van push(), pop() laat toe om meerdere transformaties uit te voeren op meerdere componenten van de sketch zonder dat ze invloed hebben op andere componenten.

### **7.3 oefeningen:**

1. 06abc\_iteration (for loop)
2. 06def\_iteration (transformations + push en pop)
3. 06\_iteration\_video (heeft nood aan local host + dom.js)

