```
<script type="application/processing">
    void spelenMet() {
        Processing;
    }
</script>
```

remie.woudt@gmail.com

Voorblad:
"Processing Logo Clipped" by Woodmath - http://www.cc.gatech.edu/grads/w/wmanzoul/portfolio_images/ProcessingLogo.png. Licensed under CC BY 3.0 via Commons - https://commons.wikimedia.org/wiki/File:Processing_Logo_Clipped.svg#/media/File:Processing_Logo_Clipped.svg

Inhoud

1 html

HTML is leuk. En zeker HTML5. Met HTML5 kun je jouw website laten leven. Kun je spellen maken voor op jouw website. En daar gaan we hier mee aan de slag. Ook al hebben we naast HTML nog wel wat meer nodig. Maar dat komt wel.

Even kort uitleggen: HTML is de taal die je gebruikt om websites te maken. En HTML5 is daar de nieuwste versie van en waarmee je dus heel veel leuke dingen kunt doen.

Een taal? Websites? We zullen even door wat droge stof moeten bijten om te leren hoe je een website maakt. Maar dat hoeft niet lang te duren. En heb je het eenmaal in de gaten dan gaan we mooie dingen maken. Webdingen!

1.1 HTML

Hoe maak je een pagina voor een website?
Een website-pagina of HTML-pagina maak je door in een programma zoals Notepad++ de tekst van jouw pagina te typen en daar een beetje code omheen te zetten. Als je daarna jouw pagina opent in een browser, dat is een programma zoals Internet Explorer, Google Chrome, Safari of, Firefox, dan zie je jouw website.

Opdracht 1

- Start het programma Notepad++ (of een andere editor)
- Als het goed is krijg je nu een leeg scherm voor je. Typ daarin de volgende tekst:

```
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <title>Mijn pagina</title>
  </head>
  <body>
    Dit wordt MIJN website!!!
  </body>
</html>
```

- Heb je het goed overgetypt? Bewaar hem dan onder de naam **`mijnpagina.html`**

Zo, jouw website is klaar. Maar nu wil je hem ook wel even zien toch?

Opdracht 2

- Start Google Chrome (of een andere browser)
- Klik op [Ctrl–O] (de toetscombinatie met de Ctrl toets en de letter O)
- Zoek **`mijnpagina.html`** en open hem

Als het goed gegaan is zie je nu in de browser de tekst die jij in **`mijnpagina.html`** getypt hebt.
Niet gelukt? Roep er iemand bij die je kan helpen!

En zo maak je dus een webpagina.
Maar waarom al die gekke tekst en haken die je in Notepad++ getypt hebt?
Die codes en haken vertellen de browser hoe jouw pagina op het scherm moet verschijnen. Maak je daar maar niet te druk om. De belangrijkste codes (we noemen ze tags) zijn:

```
<body>
</body>
```

Dit zijn de begin- en eindtags van de body. En de body is niets anders dan het gedeelte dat op het scherm verschijnt.

Opdracht 3

- Heb je enig idee hoe je een eindtag maakt? Kijk maar naar het verschil tussen de begintag en de eindtag.
- Zie je nog meer begin- en eindtags in het stukje HTML? Schrijf deze op!

Nu weet je dus hoe je een website maakt. Je neemt de codes van opdracht 1 over en zet jouw tekst tussen de begin- en eindtag van de body. Simpel hè?

1.2 Inspringen

Misschien is jou nog iets opgevallen aan de code. Heel vaak staan de begin- en eindtag recht onder elkaar. En alles wat daar tussenin komt gaat een paar spaties naar rechts.
Dat noemen we inspringen.
En inspringen is VERSCHRIKKELIJK belangrijk. Misschien niet voor zo'n kleine pagina als van opdracht 1 maar als de pagina's groter worden, kun je niet meer zonder. Want zonder inspringen wordt het een chaos!

Genoeg daarover. Laten we gaan tekenen.

1.3 Canvas en processing

Binnen HTML kunnen we een plek maken (een stukje van het scherm dus) waarop we als het ware kunnen tekenen. Dat kunnen we rechtstreeks met HTML zelf doen maar nog handiger is het om daar een hulpprogramma voor te gebruiken.

De plek om te tekenen noemen we het canvas. Maar standaard is dat tekenen nogal lastig in HTML. En daarom gebruiken we nog een andere taal die we Processing noemen.
In Processing zit een aantal functies die ons het tekenen in het canvas gemakkelijker maken. Maar daarvoor hebben we dus wel processing nodig.
Processing bestaat uit één bestand met de naam:

```
processing.min.js
```

Dit kun je vinden op **http://processingjs.org/download/**.

Ga dit bestand downloaden (klik op de rechter muisknop en kies voor **Link opslaan als…**) en plaats ze in dezelfde map als waar je ook je andere html bestand(en) hebt staan.

Ok, als je dat gedaan hebt kunnen we nu aan de slag met het tekenen van…....

2 cirkels

2.1 Een cirkel tekenen

Tekenen moet je doen dus begin maar met opdracht 4.

Opdracht 4

- Ga opnieuw naar Notepad++ en verander jouw HTML code zoals hieronder. De grijze gedeelten heb je al!

```
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <title>Mijn pagina</title>
    <script src="processing.min.js"></script>

    <script type="text/processing"
data-processing-target="canvas">
      void setup() {
        size(200, 200);
        background(100);
        stroke(255);
        ellipse(50, 50, 25, 25);
      }
    </script>

  </head>
  <body>
    <h2>
      Zo teken ik een cirkel
    </h2>
    <canvas id="canvas" width="500" height="400"></canvas>
  </body>
</html>
```

- Sla deze code weer op en bekijk het resultaat. Ziet het eruit als in afbeelding 1?

Afbeelding 1: Een cirkel met een grijze achtergrond

Nieuw zijn de twee regels die beginnen met `<script`. De eerste doet niets anders dan het bestand inlezen dat je, als het goed is, gedownload hebt. Maar het tweede script-gedeelte is de eigenlijke code. En die gaan we even doornemen.

`<script type = "text/processing" data-processing-target = "canvas">`	Dit geeft aan dat het om een stukje processing-code gaat en dat de uitvoer gaat naar het blokje met het `id="canvas"` zoals dat in de body van de pagina staat.
`void setup() { . . . . }`	Processing-code begint altijd op deze manier, met `void setup() {`. De setup methode zoals we dat noemen eindigt bij de afsluitende accolade `}`.
`size(200, 200);`	De afmetingen van het vierkant waarbinnen we gaan tekenen, hier dus 200 x 200. De eenheid hierbij is pixels of misschien duidelijker gezegd beeldschermpuntjes.
`background(100);`	De achtergrondkleur. We werken eerst alleen maar met grijstinten. 0 is daarbij zwart en 255 is wit. Een achtergrondkleur van 100 is dus een grijze achtergrond.

`stroke(255);`	De stroke is de rand rond alle figuren. stroke(255) betekent dat de rand kleur 255 heeft dus wit. Maar daar zie je niets van omdat de cirkel ook wit is. Verander de waarde van stroke eens in 0 en kijk wat er gebeurt.
`ellipse(50, 50, 25, 25);`	En hier wordt uiteindelijk de cirkel getekend. De eerste twee getallen geven de x- en y-waarde aan van het middelpunt. De beide andere getallen geven de breedte en de hoogte aan. Als de breedte en de hoogte gelijk zijn krijg je een cirkel, zo niet dan krijg je een ellips.

Dat lijkt toch niet al te moeilijk nietwaar?
Maar bij de volgende opdracht mag je er het nodige zelf aan toevoegen. Daarvoor eerst nog enkele aanwijzingen:
Met de code `fill(0)` vul je een figuur met de kleur 0 dus met zwart.
Met de code `strokeWeight(2)` wordt de rand die je met stroke tekent een stukje dikker.

Opdracht 5

- Nu je de eerste cirkel al hebt, teken je er nog wat cirkels bij tot je de beer van afbeelding 2 krijgt.
- Doe het cirkeltje voor cirkeltje. Dan kun je steeds zien of je hem op de juiste plaats krijgt.

Afbeelding 2: Een beer met Processing

2.2 Cirkels in beweging

Met processing kunnen we veel meer dan cirkels tekenen. Met name de figuur op de muis te laten reageren maakt het nog leuker. Eens kijken of dat gaat lukken met de ogen van de beer.

Opdracht 6

- Om wat beweging in de code te krijgen moeten we eerst de code een beetje anders indelen. We gaan het tekengedeelte in een aparte functie stoppen. We gaan dus het setup gedeelte en het tekengedeelte splitsen. Het script komt er dan als volgt uit te zien:

```
<script type="text/processing"
data-processing-target="canvas">

  void setup() {
    size(200, 200);
  }

  void draw() {
    // Dit tekengedeelte wordt automatisch 60 keer per
    // seconde ververst.
    // Omdat je verder in de code de opdracht fill(0)
    // gebruikt moeten we de vulkleur eerst weer op wit
    // zetten. Anders zal hij alles met zwart gaan vullen.
    fill(255);

    (hier zet je nu alle code die je gebruikt heb om de kop van opdracht 5 te tekenen)

  }
</script>
```

- Test hem nu eerst maar even uit of hij het nog doet.

De beide regels waarmee je de ogen getekend hebt zagen er ongeveer zo uit:

```
ellipse(80, 80, 4, 4);
ellipse(120, 80, 4, 4);
```

Het kan zijn dat jouw getallen een beetje anders zijn maar dat maakt niet uit.
De x-waarden van de ogen zijn in dit voorbeeld 80 en 120. Het enige wat we moeten doen is deze waarde afhankelijk te laten zijn van de bewegingen van de muis.
En dat kan bijvoorbeeld door ze te veranderen in:

```
ellipse(80+((mouseX-100)/20),80,5,5);
ellipse(120+((mouseX-100)/20),80,5,5);
```

De waarde van mouseX is de x-waarde van de plaats binnen het vierkante deel waar de muis zit. Dus als iedere 15 milliseconde de beer opnieuw getekend wordt en ondertussen de muis beweegt, wordt de x-waarde ook steeds anders. De x-waarde van de muis kan tussen 0 en 200 zijn. Door er 100 vanaf te trekken kan hij dus tussen de -100 en +100 zitten. Deel je dat door 20 dan krijgt het gedeelte tussen haakjes een waarde tussen -5 en +5. En dat is precies wat je de ogen ziet bewegen als je de muis beweegt.

Opdracht 7

- Probeer nu ook op soortgelijke manier de ogen verticaal te laten bewegen als de muis ook verticaal beweegt. (Tip: gebruik mouseY om de y-waarde van de muis te krijgen.)

- Het wordt helemaal grappig als je bij het ene oog mouseX en mouseY normaal gebruikt en die twee bij het andere oog juist omwisselt.

Afbeelding 3: Laat hem lachen!

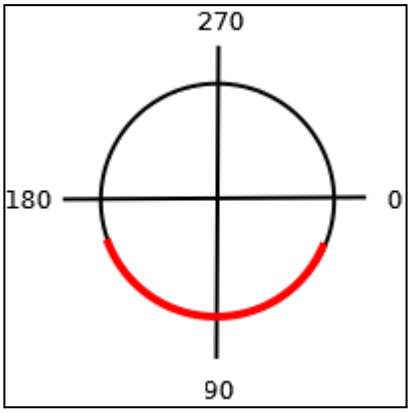
Ben je eigenlijk wel tevreden over het mondje?

Lachend zoals hiernaast is toch wel leuker.Tot nu toe hebben we ons bij het tekenen alleen nog maar bezig gehouden met ellipsen. Het mondje van afbeelding 3 is geen ellips maar een boog. In de programmeertaal noemen we dat een arc.

Eigenlijk is een arc een deel van een ellips alleen moet je begin en eind van het zichtbare deel aangeven. Dus de code lijkt op die van een ellips met twee waarden erbij. Zo dus:

```
arc(x-waarde, y-waarde, breedte, hoogte, begin, eind)
```

De eerste vier waarden zijn dus net zoals bij de ellips. Maar hoe vul je de waarden voor begin en eind in?
Die waarden geef je aan in graden of radialen. Aangezien jullie vast nog niet dagelijks met radialen werken gebruiken we hier gewoon graden.In afbeelding 4 zie je hoe binnen processing een cirkel of een ellips in graden is onderverdeeld.



Afbeelding 4: Hoe begin- en eindwaarden te bepalen voor een arc

Men begint horizontaal rechts en gaat van onderlangs naar links en dan bovenlangs weer terug. Wil je dus het rode gedeelte met arc afbeelden, dan moet die ergens beginnen bij 20 graden en eindigen bij 160.
Als je eerder het mondje hebt getekend met de volgende regel:

```
ellipse(100, 135, 35, 25);
```

Dan wordt de regel om nu zo'n boog voor het mondje te tekenen als volgt:

```
arc(x-waarde, y-waarde, breedte, hoogte, radians(20),
radians(160));
```

We moeten hier wel gebruik maken van radians bij de hoeken omdat Processing anders er van uitgaat dat de hoeken in radialen worden uitgedrukt.

Opdracht 8

- Verander nu het mondje zoals je dat eerder getekend had zodat onze beer op die van afbeelding 3 gaat lijken.

Opdracht 9

Je ziet in afbeelding 3 ook de ogen nog scheef staan omdat die reageren op de plaats van de muis. Dat kun je ook met het mondje doen.

- Vul in plaats van een vaste waarde voor de hoogte nu eens in (mouseY/8) en bekijk het resultaat.

Opdracht 10

- Doe iets soortgelijks met de neus!

3 lijnen

3.1 Een lijn heeft een begin en een eind

Met alleen maar cirkels tekenen zouden we gauw genoeg hebben van Processing. Gelukkig kan Processing veel meer maar we blijven nog wel even bij het tekenen. Vandaag worden het lijnen.
De opdracht om een lijn te tekenen is eenvoudig. Die luidt:

`line(beginXwaarde, beginYwaarde, eindeXwaarde, eindeYwaarde)`

Kortom, je tekent hier een lijn door het begin- en eindpunt vast te leggen. Ach, laten we het gewoon proberen.

Opdracht II

- Voor de afwisseling maar weer eens de volledige code. Neem deze over in een tekstbestand en bewaar dit bestand onder de naam **`lijnen.html`**

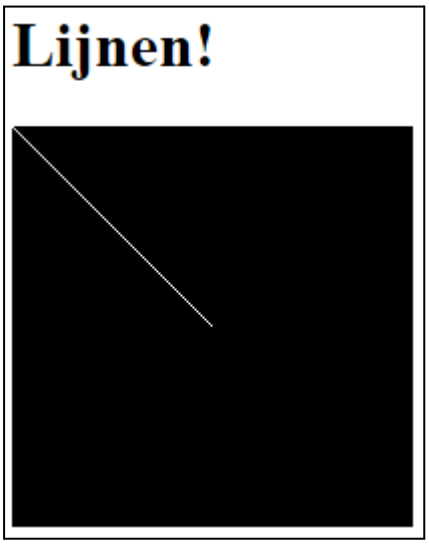
```
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <title>Lijnen</title>
    <script src="processing.min.js"></script>
    <script type="text/processing"
data-processing-target="canvas">

      void setup() {
        size(200, 200);
        stroke(255);
        strokeWeight(1);
      }
      void draw() {
        background(0);
        line(0,0,100,100);
      }

    </script>
  </head>
  <body>
    <h1>Lijnen!</h1>
    <canvas id="canvas" width="500" height="400"></canvas>
```

```
  </body>
</html>
```

- Proberen maar. In afbeelding 5 zie je hoe het eruit had moeten zien. Maar dat is vast wel gelukt.



Afbeelding 5: Een lijn

Opdracht I2

Uiteraard kunnen we hier ook weer met de lijnen gaan spelen door de muis in te zetten.

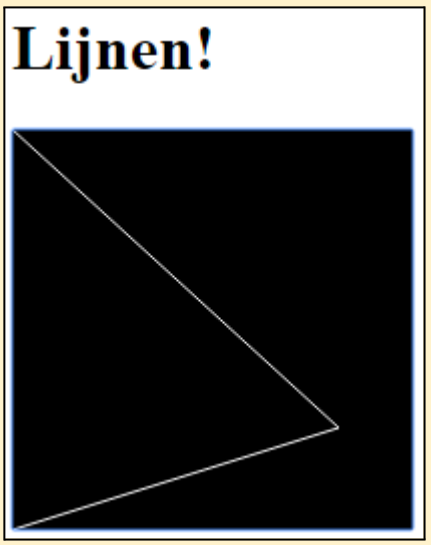
- Verander de regel van de lijn in:

```
line(0, 0, mouseX, mouseY);
```

Volgt de lijn de muisaanwijzer al?
Dat is mooi want nu kunnen we verder gaan.

Opdracht 13

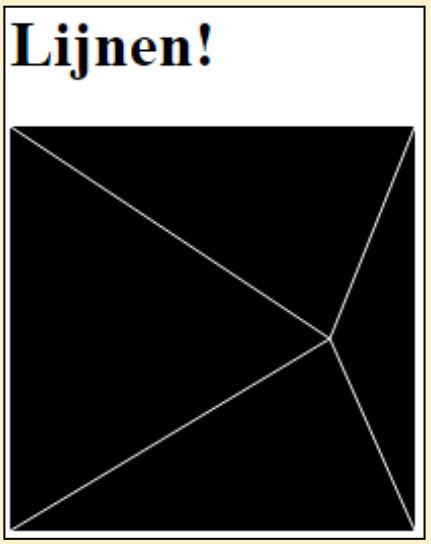
- Maak er een tweede lijntje bij en zorg dat beide lijnen de muisaanwijzer volgen. (Zie afbeelding 6).



Afbeelding 6: Een lijntje meer

Opdracht 14

- Het kan natuurlijk ook met vier lijnen! (Zie afbeelding 7).



Afbeelding 7: Vier lijnen

Opdracht 15

- Probeer de lijnen nu te tekenen vanuit het midden van de zijkanten i.p.v. vanuit de hoeken.

De kleur van de lijnen kunnen we aanpassen door vooraf de volgende opdracht toe te voegen:

```
stroke(grijswaarde);
```

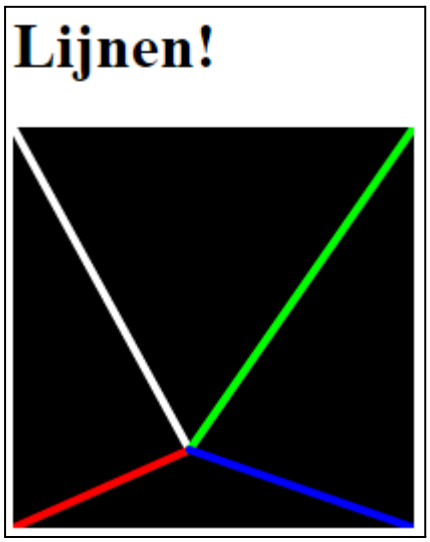
Als je dan in de plaats van grijswaarde een getal tussen 0 (voor zwart) en 255 (voor wit) in vult kun je de grijstint bepalen.

Wil je i.p.v. grijstinten echt kleuren zien, dan moet je gebruik maken van drie getallen in de volgende vorm:

```
stroke(r,g,b);
```

Daarbij staat r voor rood, g voor groen en b voor blauw. Zo zal stroke(0,255,0); een groene lijn laten zien.Probeer het maar!

Je kunt dan ook nog de dikte van de lijnen veranderen met **`strokeWeight()`**. Verander bijvoorbeeld **`strokeWeight(1)`** in **`strokeWeight(4)`**.



Afbeelding 8: In kleur en dikker

4 rechthoek

Een andere belangrijke vorm is de rechthoek. Maar met processing beslist niet moeilijk te tekenen. We beginnen weer met een stukje standaard code zoals we al eerder gemaakt hebben.

Opdracht 16

- Neem deze code over en bewaar dit bestand met de naam **rechthoek.html**

```
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <title>Rechthoek</title>
    <script src="processing.min.js"></script>
    <script type="text/processing"
data-processing-target="canvas">
      void setup() {
        background(0,0,230);
        size(400, 300);
        stroke(0);
        strokeWeight(1);
      }
      void draw() {
        fill(255,255,255);
        rect(10,10,200,150);
      }
    </script>
  </head>
  <body>
    <h1>Rechthoek!</h1>
    <canvas id="canvas"  width="500" height="400"></canvas>
  </body>
</html>
```

- Als het goed is heb je een blauwe achtergrond met daarbinnen een witte rechthoek. Bekijk je code en probeer te achterhalen waar die kleuren “gemaakt” worden.

Opdracht 17

- Maak de achtergrond groen en de rechthoek geel (met de waarden (255,255,0))

- Maak de rand van de rechthoek wit

5 robot

Inmiddels ken je nu de basisvormen van Processing. Toch niet al te moeilijk nietwaar? Daar kunnen we in de volgende opdracht eens lekker mee stoeien:

Opdracht 18

- Neem de code hieronder over en teken dan de robot zoals in afbeelding 9.

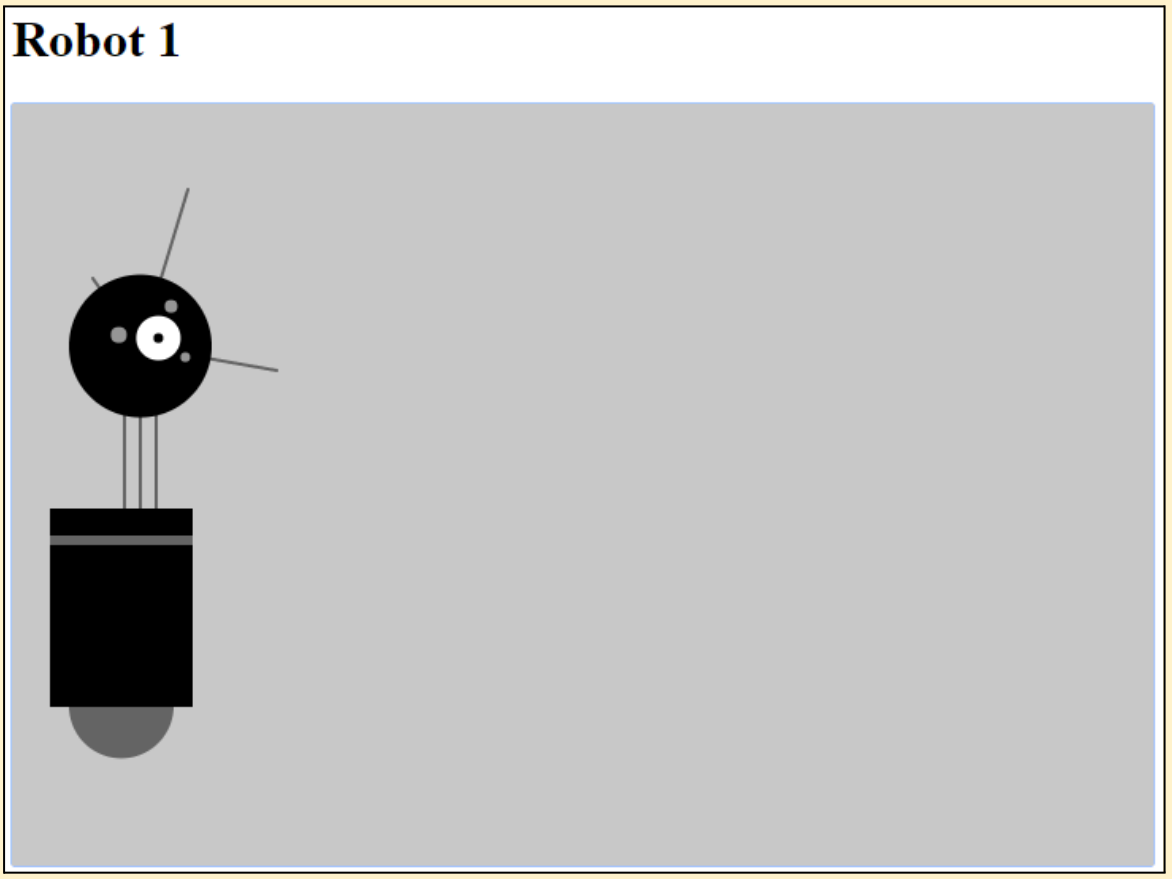
```
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <script src="processing.min.js"></script>
    <script type="text/processing"
data-processing-target="canvas">

      void setup() {
        size(720, 480);
        smooth();
        stroke(102);
        strokeWeight(2);
      }

      void draw() {
        background(204);

        //Hier komt dan de code om de robot te tekenen.

      }
    </script>
  </head>
  <body>
    <h1>
      Robot 1
    </h1>
    <canvas id="canvas" width="800" height="500"></canvas>
  </body>
</html>
```



Afbeelding 9: Robot 1

Opdracht 19

- Plaats nu bovenaan de processing code, dus vlak boven de regel
  `void setup() {`
  de volgende code:
  `int x = 10;`

- Zoek nu de laagste x-waarde op in jouw tekengedeelte. Vermoedelijk is dat de x-waarden van de buik van de robot. Onthoudt die x-waarde.

- Veronderstel nu dat die x-waarde 30 was (dat zal bij jou ongetwijfeld een andere x-waarde zijn). Zet nu achter alle x-waarden van de rechthoeken, de cirkels en de lijnen van jouw tekening de volgende code: – 30 + x (die 30 zal bij jou dus een andere waarde hebben.

- Test jouw code uit. Als het goed is, is er niets veranderd en ziet de robot er nog steeds hetzelfde uit. Is dat niet het geval dan heb je mogelijk ergens

een foutje gemaakt. Controleer je code goed.

- Wijzig nu: `int x = 10;` in `int x = 20;` en bekijk het verschil.

Als het goed gegaan is, is door het wijzigen van de waarde van x de robot een stukje opgeschoven.

Tja, dat is wel leuk, maar het kan nog leuker. Met de volgende code laten we hem door de muis bewegen:

```
if (mousePressed) {
  // x krijgt nu de waarde van de plek waar de muis is
  x = mouseX;
}
```

- Plaats deze code helemaal onderaan in het `void draw() {` gedeelte maar nog wel binnen de laatste accolade daarvan.

Als je het goed gedaan hebt kun je nu de robot verplaatsen door ergens met de muis op het scherm te klikken. Maar je kunt de robot ook met de muis ook slepen.

6 groter en kleiner

Als we de robot met de muis heen en weer kunnen laten bewegen dan moet het ook mogelijk zijn de robot te laten groeien of krimpen met de muis. Ook daarvoor hoeven we niets anders te doen dan de mouseX waarde te gebruiken bij alle coördinaten om de delen van de robot te tekenen. Maar omdat de mousX waarde kan variëren tussen 0 en 800 is dat niet een waarde die we rechtstreeks ook voor de hoogte-variaties kunnen gebruiken. Vandaar dat we de waarde gaan delen. Hieronder de code. Deze keer maar weer eens kompleet uitgeschreven.

Opdracht 20

- Neem de volgende code over en bestudeer hoe de bewegingen van de robot tot stand komen

```
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <script src="processing.min.js"></script>
    <script type="text/processing"
data-processing-target="canvas">

      int x = 10; // de x-waarde die verandert als de muis
ergens anders staat
      int h = 0; // een hoogtewaarde die afhankelijk is van
mouseX

      void setup() {
        size(720, 480);
        smooth();
        stroke(102);
        strokeWeight(2);
      }

      void draw() {
        background(204);

        // Nek
        stroke(102); // De stroke wordt kleur 102
        line(x+47, 257, x+47, 162-(h/10)); // Links
        line(x+57, 257, x+57, 162-(h/10)); // Midden
        line(x+67, 257, x+67, 162-(h/10)); // Rechts
```

```
        // Antenne
        line(x+57, 155-(h/10), x+27, 112-(h/10)); // Klein
        line(x+57, 155-(h/10), x+87, 56-(h/10)); // Groot
        line(x+57, 155-(h/10), x+143, 170-(h/10)); // Midden

        // Lijf
        noStroke(); // Geen lijn eromheen
        fill(102); // Vulkleur wordt grijs
        ellipse(x+45, 377+(h/10), 66, 66); // Het wiel
        fill(0); // Vulkleur wordt zwart
        rect(x+0, 257-(h/10), 90, 120+(h/5)); // Rechthoek
van het lijf
        fill(102); // Vulkleur wordt grijs
        rect(x+0, 274-(h/10), 90, 6); // Grijze streep

        // Hoofd
        fill(0); // Vulkleur wordt zwart
        ellipse(x+57, 155-(h/10), 90, 90); // Hoofd (cirkel)
        fill(255); // Vulkleur wordt wit
        ellipse(x+69-(x/40), 150-(h/10), 28, 28); // Grote
oog
        fill(0); // Vulkleur wordt zwart
        ellipse(x+69-(x/40), 150-(h/10), 6, 6); // Pupil
        fill(153); // Vulkleur wordt lichtgrijs
        ellipse(x+44-(x/40), 148-(h/10), 10, 10); // Kleine
oog 1
        ellipse(x+77-(x/40), 130-(h/10), 8, 8); // Kleine oog
2
        ellipse(x+86-(x/40), 162-(h/10), 6, 6); // Kleine oog
3


        if (mousePressed) {
          x = mouseX; // x kan nu een waarde krijgen van 0
tot 720
          if (mouseX > 360) {
            h = 720 - mouseX
          }
          else {
            h = mouseX
          }
        }
      }
    </script>
  </head>
```

```
  <body>
    <h1>
      Robot 1
    </h1>
    <canvas id="canvas" width="800" height="500"></canvas>
  </body>
</html>
```

En, lukt het om een beetje te volgen hoe de robot nu beweegt? We gaan nog één stapje verder.

Opdracht 21

Je ziet dat de nek van de robot niet in het midden staat. De drie lijntjes staan een beetje rechts van het midden. Ook het hoofd staat er niet midden op.

- Probeer nu de code dusdanig aan te passen dat als je de robot naar links beweegt de nek en het hoofd ook links op het lijf komt. En beweeg je hem dan weer naar rechts dan komt het hoofd en de nek weer rechts zoals hij nu is.