Introductie

1.1 Inleiding

Wat leuk dat je meedoet aan de **Q-Highschool** module **Processing**.
Er wordt veel gebruik gemaakt de video's en de website van **Daniel Shiffman**. Het is aan jezelf de bronnen die aangeleverd worden ook te gebruiken. Voor elke assignment is het de bedoeling dat je alle opdrachten maakt en inlevert voor de door de begeleider gegeven inlever datum.
Verder kan het voorkomen dat er voor bepaalde opdrachten of bronnen `[Optioneel]` staat. Dit geeft aan dat je dit materiaal niet hoeft in te leveren. Als je de stof moeilijk vindt (of meer wil weten) worden deze opdrachten sterk aangeraden. Voor sommige opdrachten staat `[Bonus]`, doe deze opdrachten als je extra verdieping en uitdaging wil.

Week 1

2.1 Leerdoelen

Deze week leer je omgaan met de volgende concepten:

- Variabelen: `int`
- Standaard functions:
  - `size(int width,int height)`
  - `circle(int x, int y, int r)`
  - `random(float bound)`
  - `stroke(float r,float g,float b)`
  - `fill(float r,float g,float b)`
  - `background(float r,float g, float b)`
  - `strokeWeight(int w)`
- Globale variablen: `width, height, mouseX, mouseY`
- Het aanroepen van functions
- Het maken van eigen functions
- Het gebruiken van for-loops
- Standaard operatoren: `a + b, a - b, a * b,a / b, a % b,a ^ b`

2.2 Uitleg

- Interactive Processing Editor
  `https://hello.processing.org/editor/`
- `[Optioneel]` Introduction - Processing Tutorial
  `https://www.youtube.com/watch?v=2VLaIr5Ckbs\&list=PLRqwX-V7Uu6ZYJC7L-r6rX6utt6wwJCyi`
- Pixels - Processing Tutorial
  `https://www.youtube.com/watch?v\=a562vsSI2Po\&list=PLRqwX-V7Uu6bsRnSEJ9tRn4V\_XCGXovs4`
- Processing Environment - Processing Tutorial
  `https://www.youtube.com/watch?v\=5N31KNgOOOg\&list=PLRqwX-V7Uu6Yo4VdQ4ZTtqRQ1AE4t\_Ep9`
- Interaction - Processing Tutorial
  `https://www.youtube.com/watch?v\=o8dffrZ86gs\&list=PLRqwX-V7Uu6by61pbhdvyEpIeymlmnXzD`
  Alleen **3.1**
- Variables - Processing Tutorial
  `https://www.youtube.com/watch?v=B\-ycSR3ntik\&list=PLRqwX-V7Uu6aFNOgoIMSbSYOkKNTo89uf`
  Alleen **4.1**

2.3 [optioneel] Voorbeelden

Bekijk de volgende voorbeelden. Ga per regel na of je snapt wat er gebeurd. Pas eventueel wat dingen aan om te kijken wat het effect is. Als je codeert mag je het internet en de voorbeelden altijd gebruiken, maak hier dus gebruik van!

1. `houses`
2. `chess`

2.4 Opdrachten

Je gaat de eerste code schrijven! Zorg ervoor je een nieuwe *sketch* aanmaakt.

2.4.1 Een cirkel

Plak de volgende code in je lege sketch. Run de sketch om te kijken wat dit doet!

```
void setup() {
    size(500,500);
    circle(100,100,70);
}
void draw() {
}
```

Pas de code aan zodat de cirkel in het midden van het scherm staat.

Opmerking

Oudere versies van Processing hebben geen `circle(x,y,radius)` functie. Gebruik dan `elipse(x,y,width,height)`

2.4.2 Het midden

Als je de grote van het scherm aanpast (pas de arguments van `size` aan), zul je zien dat de cirkel niet meer in het midden staat. Gebruik de variabelen `width` en `height` om het midden van het scherm te berekenen. Zorg ervoor dat de cirkel in het midden van het scherm blijft staan, onafhankelijk van de grote van het scherm.

2.4.3 Een sneeuwpop

Maak een sneeuwpop door twee cirkels half-boven de eerste cirkel te tekenen. Zorg ervoor dat deze cirkels ook op de juiste plek blijven staan als je de schermgrootte veranderd!

2.4.4 Een function

Het tekenen van de sneeuwpop is een simpel **algoritme**. Als je een programma wil maken is het enorm belangrijk je het op te delen. Dit opdelen gebeurd onder meer door het maken van **functions**. Plaats de volgende function **helemaal onderaan** je de sketch.

```
void drawSnowman(int x, int y) {
}
```

Plaats de volgende regel code onderaan in `setup`.

```
    drawSnowman(int(random(width)),int(random(height)));
```

Verplaats het tekenen van je sneeuwpop naar de `drawSnowman` function. Het is belangrijk dat je de sneeuwman op de coördinaten `x,y` plaatst.

2.4.5 Meer sneeuwpoppen

Maak een `for` loop in de `setup` die 10 sneeuwpoppen op willekeurige plekken op het scherm plaatst.

2.4.6 Gekleurde sneeuwpoppen

In de wereld van code kan alles! Waarom zouden we ons limiteren tot witte sneeuwpoppen? Zorg ervoor dat alle sneeuwpoppen verschillende kleuren hebben!

2.4.7 Nog meer sneeuwpoppen

Voeg de volgende function toe:

```
void mouseClicked() {
}
```

Deze function word uitgevoerd op het moment dat je met de muis op de sketch klikt. Zorg ervoor dat er een sneeuwpop op de plek van de muis verschijnt als je klikt.

Tip

Zoek op het internet naar een manier om de muis coördinaten te krijgen

2.4.8 [Extra] Hogere sneeuwmannen

Pas de function `drawSnowman` aan zodat er nog een derde variable is:

```
void drawSnowman(int x, int y, int n) {
}
```

Deze derde variable geeft de hoogte van de sneeuwpop aan. Zorg ervoor dat ook de hoogte (het aantal cirkels) variabel is! Let op! De cirkels moeten zowel kleiner worden, als boven op elkaar gestapeld worden! Zorg dat je ook 100-ballen-hoge sneeuwpoppen kunt maken! Pas vervolgens

```
drawSnowman(int(random(width)),int(random(height)));
```

aan zodat de sneeuwpoppen tussen de `2` en `6` hoog zijn.

2.4.9 [Extra] Sneeuwmannen dans

Maak de function `makeCirle`, met als parameters een `int n`

```
void makeCircle(int n) {
}
```

Deze function moet `n` sneeuwmannen in een cirkel om het midden van het scherm tekenen. De sneeuwmannen moeten allemaal op gelijke afstand van elkaar staan.

Week 2

3.1 Leerdoelen

- `x,y` coördinaten om kunnen zetten in de `PVector` objecten.
- Gebruik kunnen maken van de volgende PVector methods: `add(PVector p)`, `sub(PVector p)`, `mult(float amount)`, `rotate(float angle)`

3.2 Uitleg

- Nature Of Code Chapter 1
  `https://natureofcode.com/book/chapter-1-vectors/`
  Alleen **1.1**, **1.2**, **1.3**, **1.4**, **1.5**
- Vectors - The Nature of Code
  `https://www.youtube.com/watch?v=mWJkvxQXIa8\&list=PLRqwX-V7Uu6ZwSmtE13iJBcoI-r4y7iEc`
  Alleen **1.1**, **1.2**, **1.3**, **1.4**

3.3 Voorbeelden

1. `star`
2. `square`
3. `forest`
4. `flower`

3.4 Opdrachten

Omdat het onhandig is om telkens twee argumenten mee te moeten geven voor een positie op het scherm `int x, int y` en we een betere manier nodig hebben om met coördinaten om te gaan bestaat er in Processing de `PVector` class.

3.4.1 [optioneel] Vectoren in de wiskunde

Een vector is een verzameling van meerdere variabelen. Wij zullen ons alleen maar bezig houden met 2 dimensionale vectoren van `x,y` coördinaten. Een vector wordt als volgt genoteerd:

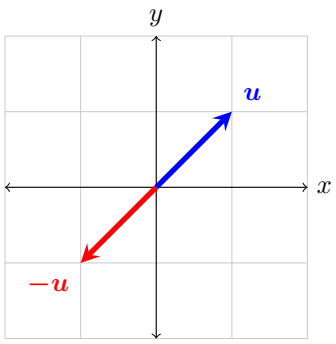
$$\vec{v} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

Er zijn een paar rekenregels, die erg voor de hand liggen als je bedenkt dat een vector gewoon een verzameling van twee coördinaten is:

$$\begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a+c \\ b+d \end{pmatrix}$$

$$a * \begin{pmatrix} b \\ c \end{pmatrix} = \begin{pmatrix} a*b \\ a*c \end{pmatrix}$$

De tweede rekenregel heet *scalaire vermenigvuldiging*. Dit geeft het uitrekken of inkrimpen van een vector weer. Dit is makkelijker te zien als we de vectoren als pijltjes (of natuurkundige krachten) tekenen:



Opdrachten

1. Teken de optelling van $\begin{pmatrix} 2 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ 3 \end{pmatrix}$
2. Bereken $2 * ((3 * \vec{a}) + \vec{b})$ met $\vec{a} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ en $\vec{b} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$
3. Bepaal het midden tussen $\vec{a}$ en $\vec{b}$. (We zoeken dus een algemene formule voor het midden tussen twee vectoren).
4. Bereken de vector op $\frac{2}{3}$ afstand tussen $\vec{a}$ en $\vec{b}$ (Wederom zoeken we dus een algemene formule).

3.4.2 [Optioneel] PVector

Processing heeft de class `PVector`, met daarin een heleboel handige methods, zie `https://processing.org/reference/PVector.html`

```
void setup() {
  PVector v1 = new PVector(3,2);
  PVector v2 = v1.copy();
  v1.add(v2);
  v2.sub(new PVector(1,1));
  v1.mult(3);
  drawDot(v1);
  drawDot(v2);
}

void drawDot(PVector v) {
  circle(v.x,v.y,5);
}
```

Opmerking

De oorsprong (0,0) zit bij computers links boven, en niet links onder zoals bij de meeste wiskundige grafieken! De y-as is als het ware gespiegeld!

Op welke coördinaten tekent dit stukje code een stip? Schrijf je antwoord in een *comment* van je sketch:

3.4.3 [Optioneel] Vectoren gebruiken

Nu je hebt geleerd wat een vector is wil je natuurlijk deze coole vectoren voor alles gebruiken! Helaas accepteren de Processing functies geen vectoren, alleen `x, y` coördinaten. Maak de volgende 3 functies af, de functie `myLine` is al gegeven.

```
void myLine(PVector v1, PVector v2) {
    line(v1.x,v1.y,v2.x,v2.y);
}

void myCircle(PVector v1, int r) {
    // TODO
}

void myTriangle(PVector v1, PVector v2, PVector v3) {
    // TODO
}
```

3.4.4 Een driehoek

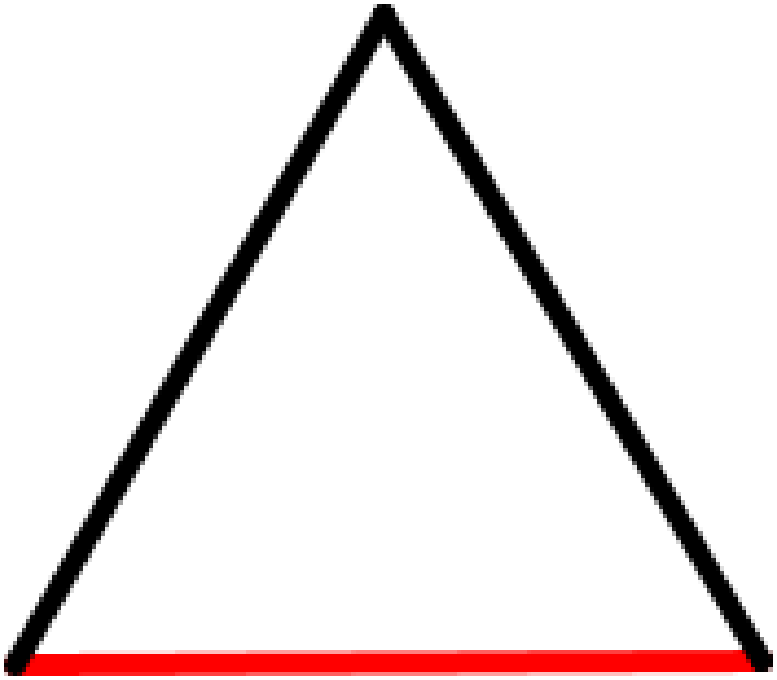
Maak de functie

```
void betterTriangle(PVector p1, PVector side) {
    // TODO
}
```

Hierbij is `p1` één van de hoekpunten en `side` één van de zijden.

Tip

Kijk naar voorbeeld `square`



/4/4

Figuur 3.1: Een triangle met de vector `side` rood gekleurd

3.4.5 Polygoon

Een gelijkzijdige polygoon of veelhoek is een figuur met `n` hoeken en lijnstukken van gelijke lengte. Voor `n = 3` is dit een *driehoek*, voor `n = 4` is dit een *vierkant*, voor `n = 5` is dit een *pentagon* en voor `n = 17` is dit een *heptadecagoon*. Maak de volgende functie:

```
void polygon(PVector center, int radius, int n) {
}
```

Deze functie moet een polygoon van `int n` hoeken tekenen met een straal van `int radius`. Maak gebruik van vectoren en gebruik de `rotate(float angle)` method.

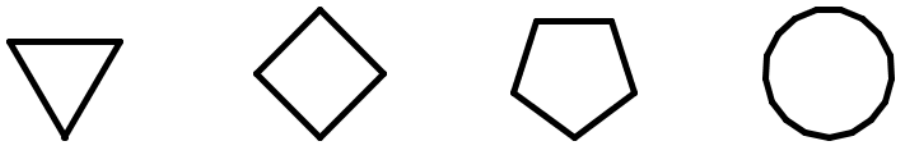
Opmerking

Een hoek wordt niet in graden uitgedrukt maar in radialen, dit betekent dat één cirkel (dus 360 graden) gelijk is aan $2 * \pi$, ofwel: `2 * PI`.

3.4.6 Middelloodlijn

Maak de volgende functie aan:

```
void bisector(PVector p1, PVector p2) {
}
```

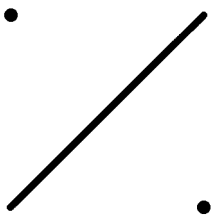


Figuur 3.2: Polygonen met n = 3, 4, 5 en 17

Deze functie moet de middelloodlijn tekenen van de twee functies met de lengte gelijk aan de afstand tussen de twee functies (zie figuur 3.3).

Tip

Probeer eerst op papier een aantal middelloodlijnen te tekenen en probeer vervolgens om de twee punten waartussen de lijn getrokken moet worden te vinden.



Figuur 3.3: Middelloodlijn, de twee stippen geven `p1` en `p2` aan

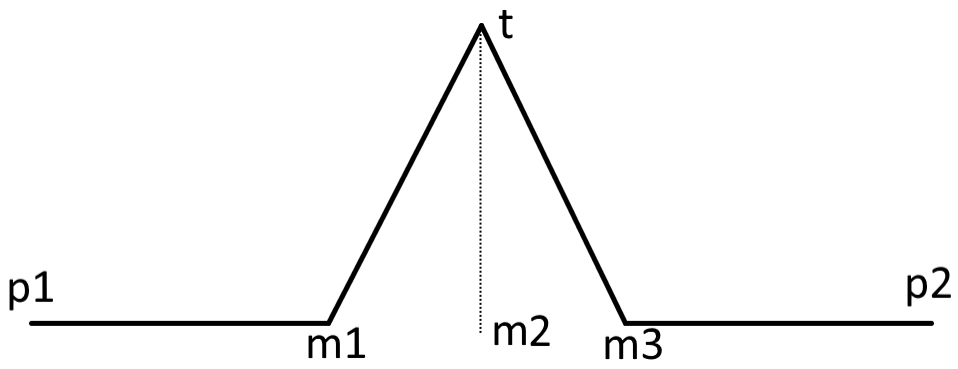
Inleveropdracht 1

4.1 Koch's Curve generatie 1

Maak de functie

```
void kochCurve(PVector p1, PVector p2) {
}
```

Deze functie tekent de volgende figuur tussen `p1` en `p2`:



Figuur 4.1: Koch's curve gen 0

Hierbij ligt `m2` midden tussen `p1` en `p2`.
De afstand tussen `p1` en `m1` is even groot als de afstand tussen `m2` en `t`.
`m1` ligt op $\frac{1}{3}$ afstand van `p1` naar `p2`.
`m3` ligt op $\frac{2}{3}$ afstand van `p1` naar `p2`.

Tip

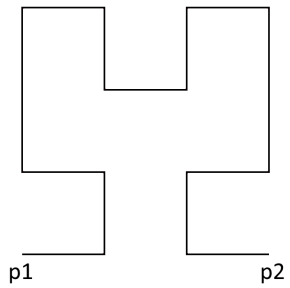
Dit is een redelijk complexe opdracht. Teken dit eerst uit, en probeer al voordat je begint met coderen te weten *wat* je precies gaat coderen. Maak gebruik van de `dist` `rotate` en `mult` PVector methods.

4.2 [Bonus] Hilbert's Curve

Maak de volgende functie:

```
void hilbertCurve(PVector p1,PVector p2) {
}
```

Die de volgende figuur (figuur 10.1) tekent:



Figuur 4.2: Hilbert's Curve

Week 3

5.1 Leerdoelen

- Begrip voor het concept recursie
- Het coderen van recursieve functies
- Het kunnen tekenen van simple fractels

5.2 Uitleg

- Fractals - The Nature of Code
  `https://www.youtube.com/watch?v=-wiverLQl1Q\&list=PLRqwX-V7Uu6bXUJvjnMWGU5SmjhI-OXef\&index=1`
  Alleen **8.1, 8.2, 8.3**
  Bij 8.3 wordt er gebruik gemaakt van een `ArrayList` voor het maken van Koch's Curve. Dit hoef je (nog) niet te kunnen.
- Chapter 8. Fractals
  `https://natureofcode.com/book/chapter-8-fractals/`
  Alleen **8.1, 8.2, 8.3, 8.4, 8.5**
  Bij 8.4 wordt er gebruik gemaakt van een `ArrayList` voor het maken van Koch's Curve. Dit hoef je (nog) niet te kunnen.

5.3 Opdrachten

5.3.1 [Optioneel] Recursieve functies

Recursieve functies zijn functies die een (makkelijkere) versie van zichzelf gebruiken voor het bereken van een antwoord. Kijk bijvoorbeeld naar:

$$f(n) = \begin{cases} 1 & \text{n} = 0 \\ 3 * \text{f(n-1)} & \text{n} > 0 \end{cases}$$

Deze functie geeft $f(n) = g^n$. Als we dit uitschrijven krijgen we: $f(4) = 3 * f(3) = 3 * 3 * f(2) = 3 * 3 * 3 * f(1) = 3 * 3 * 3 * 3 * f(0) = 3 * 3 * 3 * 3 * 1 = 81 = 3^4$

$$g(n) = \begin{cases} 1 & \text{n} = 0 \\ \text{n} * \text{g(n-1)} & \text{n} > 0 \end{cases}$$

Schrijf de uitwerking van $g(5)$ helemaal uit. Weet je ook welke functie g is?

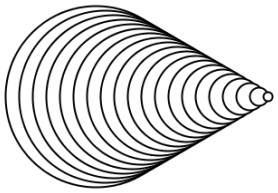
$$h(n) = \begin{cases} 1 & \text{n} = 0 \\ 1 & \text{n} = 1 \\ \text{h(n - 1) + h(n - 2)} & \text{n} > 1 \end{cases}$$

Schrijf de uitwerking van $h(4)$ helemaal uit. Weet je ook welke functie h is?

5.3.2 Recursieve circles

Als een functie *zichzelf* aanroept noemen we dit **recursie**. Je kunt met recursie allerlei coole dingen tekenen. Een recursieve functie moet wel ooit stoppen, daarom geeft je vaak een getal mee `int n` wat het aantal **itteraties** geeft, ofwel, hoe vaak de functie aangeroepen wordt, of hoe "diep"de functie gaat.
Pas de onderstaande code aan zodat het resultaat lijkt op figuur 5.1

```
void recursiveCircles(int n, PVector pos) {
  if (n > 0) {
    circle(pos.x,pos.y,n * 10);
    recursiveCircles(n - 1,pos);
  }
}
```

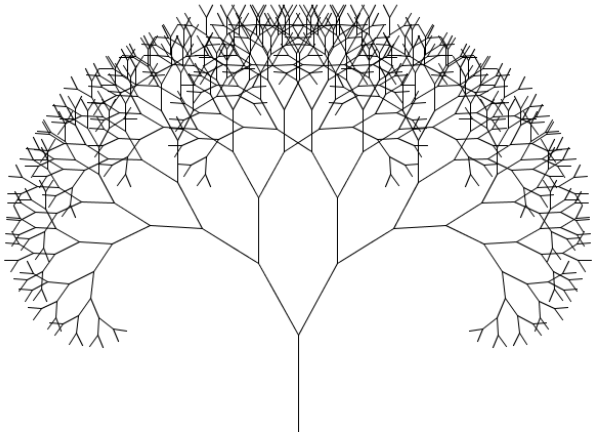


Figuur 5.1: Recursief getekende cirkels

5.3.3 Maak een binary tree

In deze opdracht ga je een **binary tree** maken (zie figuur 5.2). Een binary tree bestaat uit één lijn, genaamd de stam (trunk). Vanuit de top van deze stam "groeien" weer twee nieuwe stammen, alleen dan gedraaid onder een hoek. Maak de functie `binaryTree` af.

```
void binaryTree(int n,PVector pos,PVector trunk) {
  if (n > 0) {
    PVector tip = pos.copy().add(trunk);
    line(pos.x,pos.y,tip.x,tip.y);
  }
}
```



Figuur 5.2: Een binary tree

5.3.4 Meer soorten trees!

Vervang de hoek die je gebruikt hebt bij de vorige opdracht door `(float) mouseX / width)`. Aanschouw de epische trees!

Opmerking

Zorg ervoor dat je de functie aanroept in de `draw` functie

Week 4

6.1 Leerdoelen

- Ingewikkelde fractels kunnen tekenen
- Snappen wat een L-System is
- Extra: L-Systems kunnen gebruiken om fractels te tekenen

6.2 Uitleg

- Fractals - The Nature of Code
  `https://www.youtube.com/watch?v=-wiverLQl1Q\&list=PLRqwX-V7Uu6bXUJvjnMWGU5SmjhI-OXef\&index=1`
  Alleen **8.4, 8.5, cc#14, cc#16**

- Chapter 8. Fractals
  `https://natureofcode.com/book/chapter-8-fractals/`
  Alleen **8.6**

6.3 Opdrachten

Deze opdracht bestaat uit twee delen. Het eerste deel (het maken van een Sierpinski Triangle) hoort nog bij de stof. En het tweede deel over L-Systems is verdieping en hoef je niet te kunnen/kennen (hoewel het wel super gaaf is!).

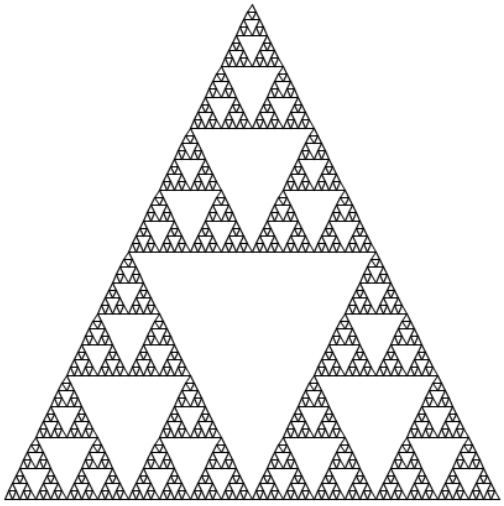
6.3.1 Sierpinski Triangle

Maak een functie die een Sierpinski Triangle tekent:

```
void sierpinski(int n, PVector p1, PVector p2, PVector p3) {
}
```

Een Sierpinski Triangle (zie figuur 6.1) bestaat uit een driehoek met hoekpunten `p1, p2, p3`. Telkens wordt het midden van de zijden bepaald. Deze middelpunten vormen samen met de orginele hoekpunten 3 kleinere driehoeken. Dit proces wordt `n` keer recursief herhaald zodat figuur 6.1 ontstaat!
Je mag gebruik maken van de volgende functie:

```
PVector midpoint(PVector p1, PVector p2) {
  return p1.copy().add(p2).mult(0.5);
}
```



Figuur 6.1: Sierpinski's Triangle met n = 5

6.3.2 [Extra] L-Systems

Lees de paragraaf over L-Systems of bekijk de video's (zie het kopje uitleg).

Eerste L-System

```
Axiom = A
Rules:
    A -> B-A-B
    B -> A+B+A
```

De eerste generatie van dit L-System geeft `B-A-B`
De tweede generatie geeft `A+B+A-B-A-B-A+b+A`
Geef zelf generatie drie.

Tweede L-Systems

```
    Axiom = F-G-G
    Rules:
        F -> F-G+F+G-F
        G -> GG
```

Geef generatie twee van dit L-System (is een boel schrijfwerk, maar het waard!)

Turtle Graphics

We gaan nu zogenaamde **turtle graphics** toepassen op het tweede L-System. Zet je pen op het papier en lees je antwoord van de vorige opdracht letter voor letter:

Als je een F tegenkomt zet een streep van 1 cm (omhoog).

Als je een G tegenkomt zet een streep van 1 cm (omhoog).

Als je een + tegenkomt draai het paoier 120 graden met de klok mee.

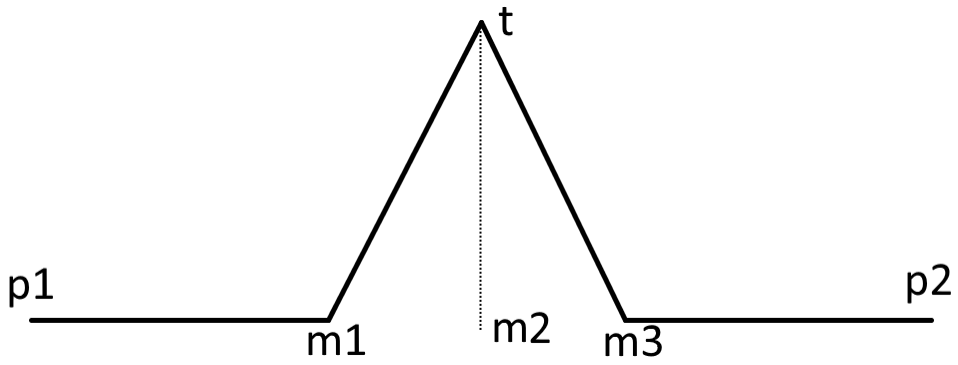
Als je een - tegenkomt draai het papier 120 graden tegen de klok in.

Als je verder wil spelen met deze L-Systems, je kunt heel veel leuke dingen op het internet vinden. Deze geweldige site:
`http://www.kevs3d.co.uk/dev/lsystems/`

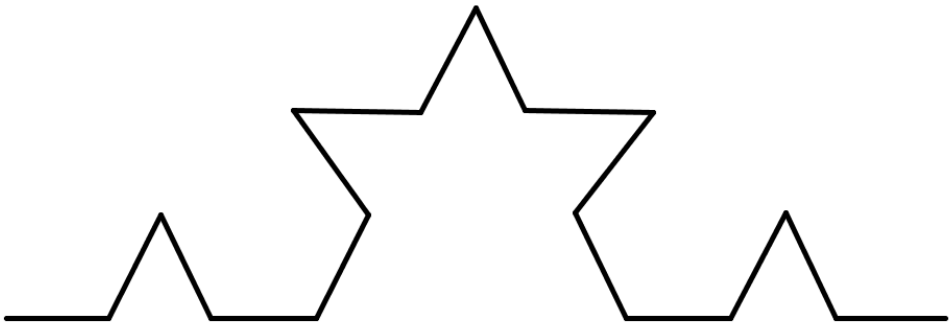
Inleveropdracht 2

7.1 Codeer Koch's Curve

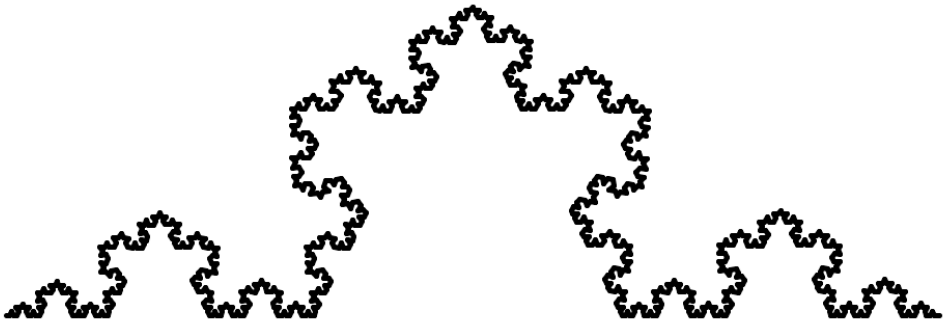
Vandaag gaan we door middel van `recursieve functies` fractals tekenen! In de vorige *assignment* heb je de functie `kochCurve` geschreven die de figuur 10.1 tekent. Als je iedere lijn vervangt door deze zelfde figuur krijg je de situatie van figuur 7.2. Als je dit oneindig vaak herhaald krijg je een **fractel** (zie figuur 7.3).



Figuur 7.1: Koch's Curve generatie 1



Figuur 7.2: Koch's Curve generatie 2



Figuur 7.3: Koch's Curve Fractel (generatie ∞)

In deze opdracht ga je een functie schrijven die dit proces `n` keer herhaald.
Maak gebruik van de al geschreven functie `kochCurve` van de vorige assignment.

```
void kochCurve(int n,PVector p1, PVector p2) {
    if (n == 0) {
        line(p1.x,p1.y,p2.x,p2.y);
    }else {
        //TODO
    }
}
```

7.2 [Bonus] Koch's Snowflake

Door Koch's curve een aantal keer te draaien ontstaat **Koch's Snowflake**.

```
void kochSnowflake(int n, int sides, int r, PVector center) {
}
```

De parameters zijn:

n het aantal itteraties van Koch's curve.

center het midden van de snowflake.

sides het aantal zijden van de snowflake.

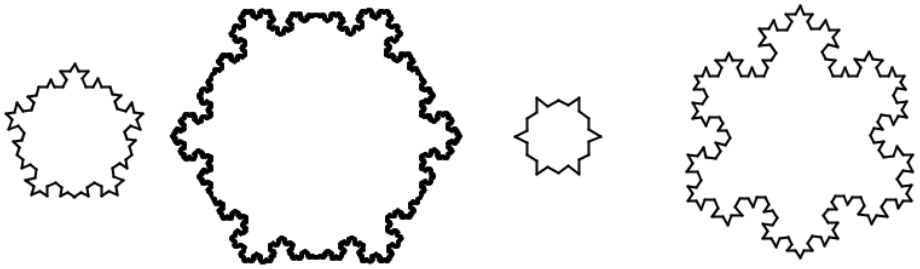
r de radius van de snowflake.

Tip

Gebruik de code die je hebt geschreven voor opdracht 2 `polygon`!

De volgende code geeft als resultaat figuur 7.4.

```
snowflake(2,5,50, new PVector(100,200));
snowflake(5,6,100,new PVector(300,200));
snowflake(1,6,30, new PVector(500,200));
snowflake(3,3,100,new PVector(700,200));
```



Figuur 7.4: Koch snowflakes!

Week 5

8.1 Leerdoelen

- Het maken en gebruiken van classes
- Het gebruiken van objecten (instances of classes)
- Het maken van methods
- Het werken met een ArrayList

8.2 Uitleg

- 8: Object-Oriented Programming
  `https://www.youtube.com/watch?v=YcbcfkLzgvs&list=PLRqwX-V7Uu6bb7z2IJaTlzwzIg_5yvL4i`
- 9: Arrays en ArrayList
  `https://www.youtube.com/watch?v=NptnmWvkbTw&list=PLRqwX-V7Uu6bO9RKxHObluh-aPgrrvb4a&index=1`
- Extra uitleg objecten
  `https://processing.org/tutorials/objects/`

8.3 Voorbeelden

- clouds
- ball

8.4 Opdrachten

Deze week gaan we voor het eerst *beweging* maken! Voortaan is het belangrijk dat je alleen maar tekent in de `draw` functie. Het is extra belangrijk dat je de uitlegvideo's over Object Georiënteerd Programmeren bekijkt!

8.4.1 Een class

Het is super handig om bepaalde variabelen en functies op die variabelen samen in één object te bundelen. Hiervoor gebruiken we een **class**. In een class kun je meerdere variabelen en functies stoppen. Zorg ervoor dat je goed snapt hoe de volgende sketch werkt:

```
Ball ball1;
Ball ball2;

void setup() {
    size(500,500);
    ball1 = new Ball(new PVector(width / 2, height / 2));
    ball2 = new Ball(new PVector(width / 2, 0));
}

void draw() {
    background(255);
    ball1.move();
    ball1.draw();
    ball2.move();
    ball2.draw();
}

class Ball {
    // Variabelen van deze class
    PVector pos = new PVector(width / 2,height / 2);
    PVector gravity = new PVector(0,5);

    // De constructor
    Ball(PVector beginPos) {
        this.pos = beginPos;
    }

    void move() {
        pos.add(gravity);
    }

    void draw() {
        circle(pos.x,pos.y,50);
    }
}
```

Pas de code aan zodat je naast de begin positie van de bal, ook de kleur kan aangeven.

8.4.2 Stuiterballen

Pas de sketch aan door het volgende toe te voegen (en de method `move` te vervangen).

```
PVector bounceForce = new PVector(0,0);

void move () {
    bounceIfBottom();
    pos.add(grivity);
    pos.add(bounceForce);
    bounceForce.mult(0.9);
}

void bounceIfBottom() {
    //TODO
}
```

Deze method moet de bal laten stuiteren als deze de onderkant van het scherm raakt.

8.4.3 Een ArrayList

In het de sketch van de vorige opdracht worden er twee variabelen gebruikt (`ball1` en `ball2`) om de ballen op te slaan. Maar wat nu als we 4 ballen op willen slaan, of 100000? We willen niet een hele lijst met variabelen maken natuurlijk. Daarom gebruiken we een `ArrayList`. Pas je gemaakte sketch aan zodat je met behulp van een `ArrayList` zoveel bellen kan maken als je wil! Pas vervolgens de code aan zodat je 100 ballen op willekeurige plekken maakt!

Tip

Je krijgt een willekeurig getal door `int(random(maximum_hoeveelheid))`

Week 6

9.1 Leerdoelen

- Omgaan met classes en objecten
- Particles maken en updaten

9.2 Uitleg

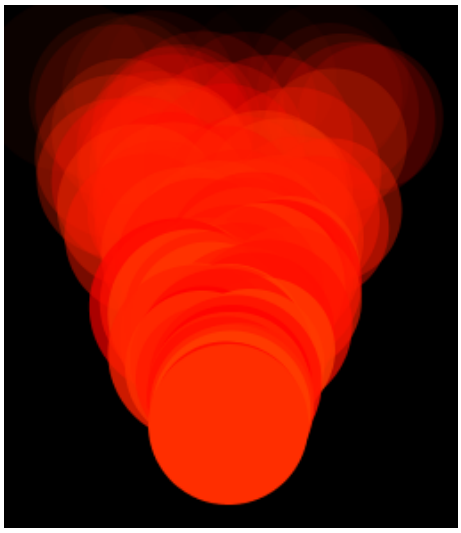
- Particle Systems - The Nature of Code
  `https://natureofcode.com/book/chapter-4-particle-systems/`
  Alleen **4.1, 4.2, 4.3, 4.4, 4.5**
- Chapter 2. Particle Systems
  `https://www.youtube.com/watch?v=vdgiqMkFygc&list=PLRqwX-V7Uu6Z9hI4mSgx2FlE5w8zvjmEy&index=1`
  Alleen **4.1, 4.2, 4.3, 4.4, 4.5**

9.3 Voorbeelden

- rain

9.4 Opdrachten

In deze opdracht ga je **vuur** maken! Dit gaan je doen door een **particle system** te maken (zie uitleg en figuur 9.1).



Figuur 9.1: Vuur!

9.4.1 Een particle

Maak een **Particle** class. Bedenk zelf welke variabelen en methods deze class waarschijnlijk moet hebben. Kijk naar figuur 9.1, welke eigenschappen heeft één vuurdeeltje?

9.4.2 Het tekenen

Maak een ArrayList van particles en zorg ervoor dat alle particles. Voeg hier één particle aan toe (in de `setup` functie). Zorg er vervolgens voor dat alle particles getekent worden (doe dit dus uiteraard in de `draw` method).

9.4.3 Het updaten

Maak een `update` method in `particle` die ervoor zorgt dat de deeltjes omhoog vliegen en een klein beetje naar links of rechts

Opmerking

Het is hier dus super handig als je gebruik maakt van een `PVector`

9.4.4 Lifetime

We willen dat één vuurdeeltje na een tijdje verdwijnt. Dit kun je doen door een variable `float lifetime = 255;` te maken. Elke update haal je hier `1` vanaf `lifetime--;`. Vervolgens moet je tijdens het updaten checken of `lifetime < 0`. Is dit het geval,

dan moet je het deeltje uit de ArrayList halen (`particles.remove(particle)`, waar `particles` de ArrayList is, en `particle` het deeltje wat je wil weghalen).

9.4.5 Kleur veranderen

Zorg ervoor dat de vuur deeltjes meer doorzichtig worden naarmate ze korter te leven hebben.

Tip

Gebruik `fill(color,alpha)` waar bijvoorbeeld `color = color(255,50,0)` en `alpha` tussen `0` en `255`

9.4.6 Vuur!

Voeg elke keer dat `draw` wordt aangeroepen een nieuwe particle toe aan de ArrayList zodat een prachtig vuur ontstaat!

9.4.7 Extra mooi vuur

Als je wil kun je ook nog extra dingen toevoegen:

- Dat de grote van de vuur deeltjes verschillend is
- Dat de kleur van de vuur deeltjes verschillend is
- Dat het vuur je muis volgt
- Dat je meerdere vuurtjes kunt maken

Inleveropdracht 3

10.1 Pong

Voor deze opdracht zul je het klassieke spel Pong gaan maken. Als je nog nooit gehoord hebt van dit spel moet je het even opzoeken om te weten waar we het over hebben (zoek op `pong game`). Je sketch moet minimaal aan de volgende eisen voldoen. Je bent verder helemaal vrij extra functionaliteit toe te voegen.

- Een bal
- Een door de gebruiker bestuurbaar batje
- Stuiter-functionaliteit

Om je een beetje te helpen staat er onderaan een stappenplan. Je hoeft dit niet te volgen, maar is wel aan te raden als je het moeilijk vind.

1. Gebruik het volgende opzetje:

```
ArrayList<Wall> walls = new ArrayList();
Wall paddle;
Ball ball;

void setup() {
  size(500, 500);
  Wall top = new Wall(0,0,width,20);
  Wall right = new Wall(width - 20,0,width - 20,height);
  Wall bottom = new Wall(0,height - 20,width,height - 20);
  paddle = new Wall(0,height / 3,20,height / 4);

  walls.add(top);
  walls.add(right);
  walls.add(bottom);
  walls.add(paddle);

  ball = new Ball();
}

void draw() {
  background(255);
  for (Wall w : walls) {
    w.draw();
    ball.bounceIfHits(w);
  }
  ball.draw();
  ball.move();
}
```

2. Maak de **Wall** class:

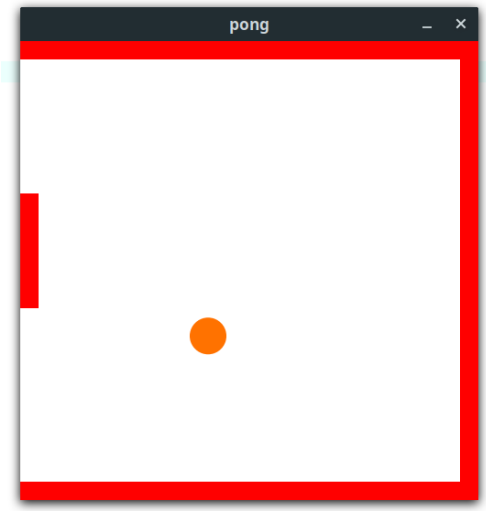
```
class Wall {
    int x;
    int y;
    int w;
    int h;

    Wall(int x, int y, int w, int h) {
      //TODO
    }

    void draw() {
      //TODO
    }
}
```

> **Tip**
>
> Test je code nu al, anders moet je straks een heleboel bugs oplossen!

3. Maak de **Ball** class.

```
class Ball {
    PVector pos;
    PVector vel;
    int radius = 20;

    Ball () {
        //TODO
    }

    void move() {
        //TODO
    }

    void bounceIfHits(Wall w) {
        //TODO
    }

    void draw() {
      //TODO
    }
}
```

4. Maak als laatste de besturing van **paddle**!



Figuur 10.1: Een voorbeeld van Pong

10.2 Bonus functionaliteit

Voeg extra features aan je sketch toe. Hier onder is een lijst met voorbeelden van extra dingen, je mag natuurlijk ook zelf iets leuks bedenken.

- Restarten
- Geef het badje een andere kleur (gebruik inheritance).
- Het bijhouden van de score (en tonen op het scherm)
- Meerdere ballen tegelijk
- Multiplayer (geef de andere speler twee andere toetsen)
- Power-ups
- Meerdere levels

Inleveropdracht 4

11.1 Maak je eigen game!

Voor deze laatste inlever opdracht ga je een eigen game maken! Leg de lat niet al te hoog, een game maken kost namelijk heel veel tijd. Ik heb onderaan een lijstje gemaakt van spellen die je na zou kunnen maken, of bedenk zelf een leuk spel!

- asteroids
- frogger
- Het dinosaurus spel van chrome
  `https://chromedino.com/`
- flappy bird
  `https://flappybird.io/`

Probeer Object Georiënteerd te programmeren. Bedenk voordat je begint met typen eerst welke objecten je nodig hebt, welke methods en variablen die objecten hebben, en wat er verder nog nodig is voor je spel. Zorg er dus voor dat je al weet wat je gaat maken voordat je begint met typen. Verder is het erg handig om wat dingen uit te tekenen, dit geeft een beter beeld van wat je wil bereiken!