# INTRO

p5.js is een open-source JavaScript-bibliotheek die creatieve technologie toegankelijk maakt voor kunstenaars, ontwerpers, studenten, docenten, en nieuwsgierige beginners - kortom: voor iedereen die wil experimenteren met code.

Met p5.js maak je in een handomdraai 2D- en 3D-animaties, werk je met video en geluid, bouw je interactieve installaties of ontwikkel je zelfs eenvoudige games. Alles draait rechtstreeks in de browser, zodat je jouw creaties meteen kunt tonen, delen en publiceren online.

Maar p5.js is meer dan een technische tool. Het is een levendige, inclusieve gemeenschap die diversiteit en toegankelijkheid centraal stelt. Waar vrouwen en mensen van kleur vaak ondervertegenwoordigd zijn in zowel de kunst- als technologiesector, zet p5.js actief in op verandering. Hier zijn verschillende perspectieven geen randzaak, maar een bron van kracht. De gemeenschap groeit dankzij de rijke mix van ideeën van iedereen die bijdraagt.

## EEN COMPUTERTEKENING MET POTLOOD

Benodigdheden:

- papier
- een potlood
- drie kleurpotloden of stiften van verschillende kleuren

Regels:

1. Teken met het potlood ergens op het papier **een driehoek**.
2. Teken, nog steeds met het potlood, **een cirkel** in de overgebleven ruimte.
3. Teken als laatste **een vierkant** in de overgebleven ruimte.
4. **Kleur** één van de vormen in met een bepaalde kleur.
5. **Kleur** één van de twee overgebleven vormen in met een andere kleur.
6. **Kleur** nu de derde en laatste vorm in met een andere kleur.

👉 Kijk zeker eens naar het project conditional design als je van regels, interactiviteit en spel in ontwerp houdt.

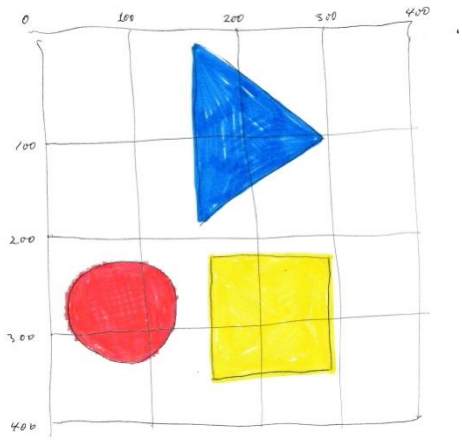
👉 ●■▲ door Bruno Munari

## VAN PAPIER NAAR CODE MET P5.JS

Voor we de tekening omzetten naar code, plaatsen we eerst enkele hulplijnen:

1. Teken met potlood een vierkant rond je drie vormen. Hoe kleiner het vierkant, hoe beter - zolang het alle vormen omsluit.
2. Trek een verticale lijn van het midden van de bovenkant naar het midden van de onderkant van het vierkant. Zo deel je de tekening in twee helften.
3. Trek een horizontale lijn van het midden van de linkerkant naar het midden van de rechterkant. Nu is de tekening verdeeld in vier kwarten.
4. Verdeel elk kwart verder in vier gelijke vierkanten. Je krijgt zo een raster van 16 vierkanten.
5. Plaats afmetingen bij de lijnen. Linksboven is punt 0. De uiterst rechtse verticale lijn is 400, net als de onderste horizontale lijn. De tussenliggende lijnen liggen op 100, 200 en 300 - zowel verticaal als horizontaal.

Je schets moet er ongeveer zo uitzien:




Deze toolkit bevat in het volgende deel een beknopte p5.js-tutorial met de titel ▲■● (TRC).

De workshop biedt een snelle en toegankelijke kennismaking met programmeren in JavaScript. Door haar beknoptheid worden bewust heel wat aspecten niet behandeld.

Aanvullend vind je op de p5.js-website een Tutorials sectie met Introduction to p5.js-lessen. Wie liever met een boek aan de slag gaat, kan ik het compacte Getting Started with p5.js van Lauren McCarthy, Casey Reas en Ben Fry aanbevelen.

Achteraan dit document vind je het hoofdstuk Verder leren en lezen, met extra links rond generatieve kunst en creative coding.

# ▲■● TRIANGLE, SQUARE, CIRCLE

👉 ▲■● code collection: https://editor.p5js.org/hendrikleper/collections/HyPmDVNJf

## KENNISMAKING MET DE P5.JS EDITOR

👉 https://editor.p5js.org/

### user interface

1. **Menu bar**: gebruikt om schetsen op te slaan, te openen, te delen en te downloaden.
2. **Control bar**: bevat de knoppen **Start** & **Stop** om de schets te bedienen.
3. **Edit pane**: gebruikt om de p5.js-statements in te voeren die een sketch vormen.
4. **Console pane**: gebruikt om status- en systeem**berichten** weer te geven.
5. **Preview pane**: gebruikt om het **visuele resultaat** van de sketch weer te geven.

### maak een account aan

### new sketch

Een programma in p5.js noemen we een schets.

Extra: auto-refresh, p5.js versies & preferences

## FUNCTIES

### function setup() en draw()

Elke sketch bevat twee kernfuncties:

- setup()
  Wordt één keer uitgevoerd bij het starten van de sketch.
  Ideaal voor het instellen van canvasgrootte, kleuren, variabelen, enz.
- draw()
  Wordt herhaaldelijk uitgevoerd (standaard aan 60 frames per seconde), zolang de sketch actief is. Hierin gebeurt de animatie of herhaalde logica.

**Syntax en structuur:**

- Functienamen worden gevolgd door haakjes `()` waartussen eventueel parameters staan.
- De inhoud wordt gegroepeerd tussen accolades `{ … }`. Dit noemen we een **codeblock**.

### createCanvas()

Maakt een canvas van 400 bij 400 pixels.

Een pixel is het kleinste beeldpunt en vormt de eenheid van het canvas.

Syntax en naamgeving:

- Elke functie wordt afgesloten met een `;` (puntkomma of semicolon).
- Functienamen die uit twee woorden bestaan, zoals `createCanvas()`, gebruiken **camelCase** (een schrijfwijze waarbij samengestelde woorden aan elkaar staan en elk nieuw woord met een hoofdletter begint, behalve het eerste.)

### background(220)

wat zou dit kunnen zijn?

## P5.JS ONLINE REFERENCE

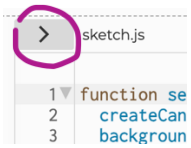
De p5js reference geeft je eenvoudige uitleg over de verschillende onderdelen van de code.

👉 https://p5js.org/reference/

## P5 ANATOMIE: HTML, CSS & JS

Een p5.js schets bestaat eigenlijk uit 3 documenten: een HTML pagina, CSS stylesheet en JavaScript document. Naast sketch.js kan je ook de index.html en style.css bekijken of aanpassen.

👉 https://editor.p5js.org/hendrikleper/sketches/4gUV95rnv




## FIRST STEPS 🦶 TRIANGLE SQUARE CIRCLE

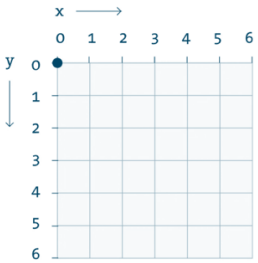
👉 https://editor.p5js.org/hendrikleper/sketches/DTEKcEQqx

### triangle() square() circle()

Zolang onze schets statisch blijft kunnen we alles toevoegen aan de setup functie.

### coördinaten

Het canvas heeft een coördinatensysteem met in de linkerbovenhoek het punt 0,0 of 0 op de horizontale X as en 0 op verticale Y as.




## CANVAS COLOR & ORDER

👉 https://editor.p5js.org/hendrikleper/sketches/lC4nnEXHN

**Kleurfuncties in P5.js**

- **background()**, **fill()**, **stroke()**
  Bepalen respectievelijk de achtergrondkleur, vulkleur en lijnkleur van vormen.
- **noFill()**, **noStroke()**
  Zorgen ervoor dat vormen geen invulling krijgen (noFill) of geen contouren (noStroke).

- **strokeWeight()**
  Bepaalt de dikte van lijnen, punten en contouren van vormen.

**Kleuren noteren**

- Kleuren worden meestal genoteerd als **(R, G, B)** met waarden van **0 tot 255**.
- Je kunt ook **(R, G, B, A)** gebruiken, waarbij **A** staat voor *alpha* (transparantie):
- 0 = volledig transparant
- 255 = volledig dekkend

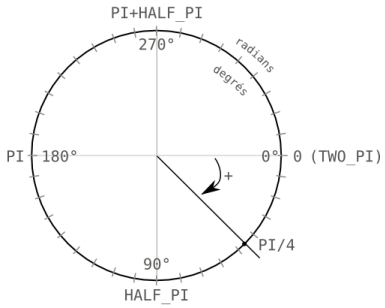
👉 Voor intuïtieve kleurkeuze: gebruik de Google Color Picker (of de nieuwe color picker in de editor op te roepen via o.a. string kleurkeuze, bijv. 'red')

## andere vormen

**Vormen tekenen in P5.js**

- point(x, y) tekent een punt op coördinaat (x, y)
- line(x1, y1, x2, y2) tekent een lijn van (x1, y1) naar (x2, y2)
- rect(x, y, w, h) tekent een rechthoek waarbij (x, y) de linkerbovenhoek is met breedte (w) en hoogte (h)
- ellipse(x, y, w, h) tekent een ellips gecentreerd op (x, y), (w) en (h) zijn breedte en hoogte
- arc(x, y, w, h, start, stop) tekent een boog binnen een ellipsvorm waarbij start en stop de hoeken instellen waarbinnen de boog getekend wordt, steeds met de klok mee. Start en stop worden radialen opgegeven.
  Een zevende parameter, mode, is optioneel en bepaalt de vulstijl: OPEN (halve boog), CHORD (gesloten halve boog) of PIE (taartpunt).

*Radians of Radialen zijn een meeteenheid voor hoeken in de wiskunde en natuurkunde, waarbij een volle cirkel gelijk is aan 2π radialen (ongeveer 6,28 radialen) in plaats van 360 graden.*




**Complexere vormen**

- Gebruik vertex(x, y) om hoekpunten van een vorm te definiëren.
- Deze worden geplaatst tussen beginShape() en endShape().

## Volgorde

De volgorde waarin functies aan een sketch worden toegevoegd is belangrijk, omdat ze één voor één na elkaar (en soms ook bovenop elkaar) worden getekend.

# RANDOM VARIABLE PRINT

👉 https://editor.p5js.org/hendrikleper/sketches/2poyTZXh5

## Getallen en willekeur

**random()** geeft een willekeurig (decimaal) getal terug binnen een opgegeven bereik, bijv. random(0, 100).

**int vs float**
int of integer staat voor een geheel getal, bijv. 0, -3, 42, 1000
float of floating point number is een decimaal getal, bijv. 3.14, -0.5, 100.0, 2.718

**int()** en **floor()**
Worden gebruikt om decimale getallen af te ronden naar gehele getallen:

- int() rondt af naar het dichtstbijzijnde geheel getal
- floor() rondt altijd naar beneden af

In p5.js kun je **functies aanroepen binnen andere functies**, bijv `circle(x,y,int(random(50,100)));`
In dit geval wordt eerst random(50, 100) uitgevoerd, daarna wordt het resultaat omgezet naar een geheel getal met int(), en dat getal wordt vervolgens gebruikt als diameter in de circle() functie.

Het correct plaatsen en sluiten van haakjes is hierbij heel belangrijk. Zeker wanneer meerdere functies genest zijn (dus binnen elkaar worden aangeroepen) kan het snel fout gaan. Een vergeten of verkeerd geplaatst haakje leidt vaak tot fouten zoals: `SyntaxError: missing ) after argument list`

## Variabelen

Een variabele is als een doos met een label: je stopt er iets in (zoals een getal, tekst of kleur), en je kunt die waarde later weer gebruiken of veranderen.
Een variabele wordt **gedeclareerd** met het sleutelwoord **let**, bijvoorbeeld: let x = 0;

Dit bestaat uit:

1. let - letterlijk: “laat x zijn 0”
2. x is een naam die je zelf kiest, het kan dus ook *emmer* zijn
3. Een is-gelijkteken (=) om een waarde toe te kennen
4. De waarde die je wilt opslaan

**Datatypes**
Wij willen in de variabele x een getal opslaan, maar er bestaan verschillende datatypes. Een datatype bepaalt welk soort informatie een variabele bevat. Drie veelvoorkomende types in p5.js zijn:

- Number: getallen
- String: tekst tussen aanhalingstekens, zoals "hallo"
- Boolean: true of false, voor waar/niet waar

**Globale vs lokale variabelen**

- Globale variabelen: gedeclareerd buiten functies, overal toegankelijk
- Lokale variabelen: gedeclareerd binnen een functie of blok, alleen daar beschikbaar

## Debuggen

print() is handig om waarden van variabelen te tonen in de console tijdens het debuggen.

# TIJD VOOR BEWEGING

👉 https://editor.p5js.org/hendrikleper/sketches/kjMbUuj4V

We verhuizen onze code of functies naar de draw-loop. Hierdoor genereert het een compositie die bij elke frame zal veranderen.

### frameRate()

Stelt het aantal beelden in dat per seconde getekend moet worden, bijv. frameRate(5)

### noLoop()

Stopt de code in draw()

## MUISINTERACTIE

👉 https://editor.p5js.org/hendrikleper/sketches/5Y-MIHzkn

### Interactie met de muis

mouseX & mouseY zijn **systemvariabelen**, variabelen die automatisch worden bijgehouden door p5.js. Ze geven **de huidige positie van de muis** op het canvas:

- mouseX = horizontale positie (x-coördinaat)
- mouseY = verticale positie (y-coördinaat)

### width & height

width en height zijn ook systemvariabelen, ze houden de breedte en hoogte van het canvas in pixels bij.

### rectMode(CENTER)

De functie rectMode(CENTER) verandert de manier waarop rechthoeken worden getekend met rect(). Normaal gesproken begint rect(x, y, w, h) te tekenen vanuit de **linkerbovenhoek** van de rechthoek. Maar zodra je rectMode(CENTER) gebruikt, wordt het **middelpunt** van de rechthoek als startpunt genomen. De waarden x en y geven dan het **centrum** van de rechthoek aan.

### mousePressed()

De functie **mousePressed()** wordt automatisch uitgevoerd telkens je met de muis op het canvas klikt. Je gebruikt deze functie om iets te laten gebeuren bij een muisklik, zoals een tekening maken of een kleur veranderen.

Functies zoals **mousePressed()** en **keyPressed()** heten **eventhandlers**: ze reageren op gebeurtenissen (events) en horen **niet in de draw()-loop**, omdat p5.js ze zelf aanroept wanneer het event plaatsvindt.

## LINEAIRE BEWEGING & CONDITIONAL STATEMENTS

👉 https://editor.p5js.org/hendrikleper/sketches/FW5wWIrhl

### Lineaire beweging

De beweging van een object dat zich gelijkmatig in één richting verplaatst (met een constante snelheid).

We gebruiken een variabele let x = 0; om de positie van het object bij te houden, en verhogen (of verlagen) die waarde telkens in de draw()-functie x = x + 1;

Om te voorkomen dat vormen uit het canvas verdwijnen, kun je grenzen instellen zodat ze binnen het canvas blijven. Dit kunnen we doen met constrain() of een voorwaardelijke controle.

### constrain()

```
x = constrain(x, 50, width-50);
```

De functie constrain() zorgt ervoor dat x nooit kleiner wordt dan 50 of groter dan width-50.

## conditional statements

of een IF statement.

**Voorwaardelijke verklaringen** bepalen wanneer specifieke regels code worden uitgevoerd. Deze test zorgt ervoor dat onze vormen binnen het canvas blijven.

```
if (x > width+70) {
  x = -70;
}
```

Dus als de x-positie van een vorm groter wordt dan de canvasbreedte, dan wordt x opnieuw ingesteld op -70. Hierdoor start de vorm net buiten beeld aan de linkerkant, en schuift dan weer het canvas in.

## Operators

Het groter-dan teken of `>` is een **operator**. Operators zijn **symbolen die bewerkingen uitvoeren op waarden of variabelen**. Ze zijn essentieel in conditionele statements zoals if, omdat ze helpen bij het vergelijken en berekenen.

Veelgebruikte vergelijkingsoperators:

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

## extra: bouncing

👉 https://editor.p5js.org/hendrikleper/sketches/SoS2oHhjl

```
if (x > width - radius || x < radius) {
  xspeed = xspeed * -1;
}
```

Deze test zorgt ervoor dat de vorm binnen het canvas blijft door de beweging aan te passen zodra een grens wordt bereikt.

De twee pipes **||** staan voor **de logische OR-operator**. Deze geeft als uitkomst true als aan één of meer van de voorwaarden voldaan is. Nog 2 andere logische operatoren zijn **&&** (AND) en **!** (NOT).

`&&` (AND) beide voorwaarden moeten waar zijn
`||` (OR) minstens één voorwaarde moet waar zijn
`!` (NOT) keert een voorwaarde om (waar wordt onwaar)

**De** `* -1` **in de regel** `xspeed = xspeed * -1;` **zorgt ervoor dat de richting van de snelheid omkeert.**
Als xspeed bijvoorbeeld **positief** is (zoals 2), dan wordt het **negatief** (-2), waardoor het object in de tegenovergestelde richting beweegt.

De eventhandler **keyPressed()** + code zorgt ervoor dat wanneer je de "f"-toets indrukt de sketch schakelt tussen fullscreen en normale weergave. Als fullscreen wordt geactiveerd, verdwijnt de muiscursor.

De eventhandler **windowResized()** + functie resizeCanvas zorgt ervoor dat het canvas zich aanpast aan de grootte van het venster wanneer je het vergroot of verkleint. Zo blijft de sketch altijd zichtbaar en correct geschaald.

## CIRCULAIRE BEWEGING

👉 https://editor.p5js.org/hendrikleper/sketches/7aTz1veWP

### sin() & cos()

De functies **cos(angle)** en **sin(angle)** berekenen de x- en y-coördinaten van een punt op een cirkel met een bepaalde **hoek** (angle). Vaak zal je ook de naam *theta* terugvinden voor een hoekbepaling.

Stel je een cirkel voor met het middelpunt op (0,0). Als je een bepaalde hoek (angle) neemt:

- cos(angle) vertelt je **hoe ver naar links of rechts** het punt ligt.
- sin(angle) vertelt je **hoe ver omhoog of omlaag** het punt ligt.

```
x = cos(angle) * radius;
y = sin(angle) * radius;
```

In het bovenstaande voorbeeld staat **radius** bepaalt hoe ver het punt van het middelpunt van de cirkel ligt

```
angle += speed;
```

Deze regel verhoogt de hoek telkens met een beetje (speed), waardoor het punt rond de cirkel beweegt.

### map()

map() herrekent een waarde van een bepaald bereik naar een ander bereik. Je kan het zien als **een herschaling**: een waarde die bijvoorbeeld tussen 0 en 100 ligt, wordt omgerekend naar een evenredige waarde tussen bijvoorbeeld 0.01 en 2.0

## WHILE LOOP

👉 https://editor.p5js.org/hendrikleper/sketches/hFpRanscn

Een while-loop **herhaalt een blok code zolang aan een bepaalde voorwaarde voldaan is**.

```
while (x < width) {
  circle(x, height / 2 + random(-mouseX / 10, mouseX / 10), diam);
  x = x + 100;
}
```

In 4 stappen:

1. Voorwaarde controleren: is `x < width` ?
2. Code uitvoeren: als dat zo is, teken een cirkel op positie x.
3. Waarde aanpassen: verhoogt x met 100, zodat de volgende cirkel verder naar rechts komt.
4. Herhalen: de loop gaat terug naar stap 1 en herhaalt dit totdat x niet meer kleiner is dan width.

De while-loop zorgt ervoor dat **meerdere cirkels naast elkaar** worden getekend, met een vaste afstand van 100 pixels tussen elke cirkel.

## FOR LOOP

👉 https://editor.p5js.org/hendrikleper/sketches/JKoVDNNPdo

Een for-loop is een manier om een stuk code meerdere keren uit te voeren, telkens met een andere waarde. Het is een erg handige en snelle manier om patronen te tekenen.

```
for (let i = 0; i < width; i += gridsize) {
  // Code binnen deze blok wordt herhaald
}
```

Wat gebeurt er hier?

1. Initialisatie: `let i = 0`
   De variabele i begint bij 0. Dit is de startpositie op de x-as.
2. Voorwaarde: `i < width`
   Zolang i kleiner is dan de breedte van het canvas (width), blijft de loop doorgaan.
3. Update: `i += gridsize`
   Na elke herhaling wordt i verhoogd met gridsize. Zo springt de loop telkens een stukje verder op de x-as.

## NESTED FOR LOOPS

👉 https://editor.p5js.org/hendrikleper/sketches/xqq54a8w-

Een **nested for-loop** is een **for-loop binnen een andere for-loop**. Dit wordt vaak gebruikt om **tweedimensionale structuren** te maken, zoals een raster van vormen op een canvas.

Een voorbeeld met een geneste structuur:

```
let gridsize = 100;
let diam = 40;

function setup() {
  createCanvas(600, 600);
  background(0);
  for (let x = 0; x < width; x += gridsize) {
    for (let y = 0; y < height; y += gridsize) {
      stroke(255);
      line(x, 0, x, height); // verticale lijnen
      line(0, y, width, y);  // horizontale lijnen
      noStroke();
      fill(255, 0, 0);
      circle(x + gridsize / 2, y + gridsize / 2, diam); // cirkel in elk vak
    }
  }
}
```

Hoe werkt dit?

- Buitenste loop (x): loopt over de breedte van het canvas.
- Binnenste loop (y): voor elke x-waarde, loopt deze over de hoogte van het canvas.
- Dus voor elke kolom (x) worden alle rijen (y) doorlopen. Hierdoor wordt de code binnenin uitgevoerd voor elke combinatie van x en y, wat resulteert in een grid van cirkels.

Stel je een schaakbord voor:

- De buitenste loop gaat over de kolommen (van links naar rechts).
- De binnenste loop gaat over de rijen (van boven naar beneden).
- Samen vullen ze elk vakje met een vorm.

## TRANSFORMATIES

👉 https://editor.p5js.org/hendrikleper/sketches/euqF426_V

### translate(x, y)

Verplaatst het **nulpunt van het canvas** naar een nieuwe positie. Alle vormen die je daarna tekent, worden relatief ten opzichte van die nieuwe oorsprong geplaatst.
Handig voor het positioneren van objecten zonder telkens coördinaten te herberekenen.

```
translate(100, 50);
circle(0, 0, 30); // wordt getekend op (100, 50)
```

### rotate(angle)

Draait het canvas rond het huidige nulpunt (dat je eventueel met translate() hebt verplaatst). De rotatie gebeurt **in radialen**.

```
rotate(PI / 4); // draait 45 graden
```

### angleMode(DEGREES)

Zet de hoekeenheid om van **radialen** naar **graden**. Maakt het makkelijker om met hoeken te werken als je gewend bent aan graden.

```
angleMode(DEGREES);
rotate(45); // draait 45 graden
```

### push() & pop()

**push()** slaat de huidige tekeninstellingen op (zoals translate, rotate, fill, enz.).
**pop()** herstelt die instellingen naar wat ze waren bij de laatste push().

Dit is superhandig om lokale transformaties toe te passen zonder het hele canvas te beïnvloeden.

```
push();
translate(100, 100);
rotate(45);
rect(0, 0, 50, 50); // getekend en geroteerd
pop();
rect(0, 0, 50, 50); // getekend zonder transformatie
```

## ARRAY

👉 https://editor.p5js.org/hendrikleper/sketches/C-uWy67O7

Een array is **een lijst van gegevens** die je in één variabele kunt opslaan zoals getallen, tekst of objecten. Je kan het bekijken als een doos met vakjes, waarbij elk vakje een waarde bevat. Je kunt elk vakje bereiken via een indexnummer, beginnend bij 0.

In dit voorbeeld gebruiken we een array genaamd *vormen* om drie verschillende vormtypes op te slaan: "driehoek", "cirkel" en "vierkant" (3 keer een string-datatype). Telkens als je op de muis klikt, kiest het programma willekeurig één van deze vormen uit de array met random(vormen) en tekent die op een willekeurige plek op het canvas.

## SAVE, SHARE & EXPORT

### Save

Zorg dat je bent ingelogd, anders kun je niet opslaan.

### Share

Zodra je project is opgeslagen, kun je het delen via Bestand > Delen.

### export as png, gif & svg

👉 https://editor.p5js.org/hendrikleper/sketches/iGnY1gcWg

### export as gif

We gebruiken de p5.createloop library, ontwikkeld om **naadloze, herhalende animaties** te maken. Vooral handig voor het exporteren van GIFs dus.

Voeg eerst p5.createloop.js toe aan je project in index.html.

animLoop.theta *is een hoekwaarde die tijdens de animatie vloeiend oploopt van 0 tot 360° (of van 0 tot TWO_PI). Theta bepaalt dus de positie op een cirkel en zorgt voor vloeiende, herhalende circulaire beweging.*

👉 https://editor.p5js.org/hendrikleper/sketches/nMRdbAvBP

### export as svg voor penplotters

Hier gebruiken we p5.plotSvg library voor het exporteren van SVG-bestanden, speciaal voor **penplotters**.

*Let op: p5.plotSvg is geen algemene SVG-bibliotheek. Het focust enkel op de geometrie van paden, niet op hun visuele stijl, omdat de uiteindelijke weergave afhangt van het fysieke tekengereedschap dat je gebruikt.*

Voeg eerst p5.plotSvg.js toe aan je project in index.html.

👉 https://editor.p5js.org/hendrikleper/sketches/sWngrT0Cm

## BONUS

### interface sliders & buttons

👉 https://editor.p5js.org/hendrikleper/sketches/k61yOtmzZ

### simple dom elements

👉 https://editor.p5js.org/hendrikleper/sketches/VYqBEB2JR

# VERDER LEREN & LEZEN

Meer links m.b.t. generatieve kunst & creative coding

## VIDEO'S EN LESSEN

*The Coding Train* P5js track met Daniel Shiffman

The Anatomy of a p5js Sketch - A tutorial for new coders van Paul Wheeler

Six Features of Programming Languages - demonstrated in p5.js van n1ckfg

## BOEKEN

Aesthetic Programming: A Handbook of Software Studies door Winnie Soon & Geoff Cox

*Generative Design* door Benedikt Groß

## COMMUNITIES

*Open Processing* is een online platform en community gericht op het delen en creëren van digitale kunst, interactieve schetsen en creatieve codeerprojecten

## GENERATIVE & COMPUTER ART HISTORY

The ReCode Project, an active archive of computer art is a community-driven effort to preserve computer art by translating it into a modern programming language.

Generative Art Timeline Ontdek de 70.000-jarige geschiedenis van generatieve kunst door middel van honderden mijlpalen en tien hoofdstukken (door Peter Bauman)

Computer Grrrls Women & machines timeline door Marie Lechner & Inke Arns

A Longer History of Generative Art Nick Lambert, een van 's werelds meest vooraanstaande wetenschappers op het gebied van computerkunst, schetst de geschiedenis van generatieve esthetiek tot aan NFT's.

## GENERATIVE ART BLOGS

*Generative Artistry* Tutorials & Podcast

*Tyler Hobbs*

*Matt DesLauriers*

## MATH

*Sine / Cosine Reference*

*Sine and Cosine Calculator*

*Linear Interpolation* — Introduction to lerp

## VERWANTE KUNSTBOEKEN EN ZINES

*Circle, Square, Triangle* by Bruno Munari

*The ABCs of Triangle, Square, Circle: The Bauhaus and Design Theory* by Ellen Lupton