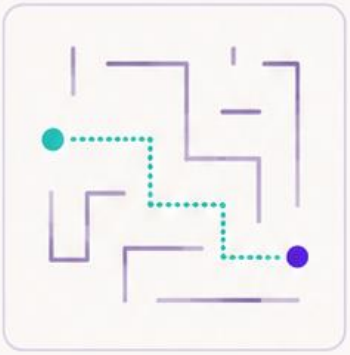
Blockly in de Klas

Van computationeel denken naar programmeren

- **computationeel** denken
- **algoritmes** ontwikkelen
- **programmeerconcepten** gebruiken
- van **blokken** naar **javascript**



01

BEGRIJP HET PROBLEEM

02

BOUW MET BLOKKEN

```
1  while (notDone()) {
2    moveForward();
3    turnRight();
4    moveForward();
5  }
```

03

SCHRIJF HET IN JAVASCRIPT

NAAM

KLAS

Inhoudsopgave

Inleiding:
Van computationeel denken naar programmeren

Je hebt al geleerd over computationeel denken. Dat betekent:

Definitie:

> Een probleem zo bekijken dat je het kunt opdelen, onderzoeken, omzetten naar duidelijke stappen en een algoritme kan ontwerpen om dat probleem op te lossen.

In deze cursus gebruiken we die denkstappen om te leren programmeren. Het ontwikkelen van die computeralgoritmes. Eerst gaan we aan de slag met grafische blokken. Daarna met JavaScript-code.

Duidelijk versus ambigu

Een mens is een heel talig wezen. Een mens begrijpt vaak wat je bedoelt als je een commando of taak geeft, ook als jouw opdracht niet helemaal duidelijk is. Een computer kan dat (doorgaans) niet. Volgende opdrachten zijn te vaag voor een computer. Mensentaal is met andere woorden te **ambigu** voor een computer.

Voorbeeld:

```
Zet het afval buiten.
Neem het fruitsap uit de koelkast en controleer even de datum.
Smeer jouw boterham met chocopasta.
```

Neem het eerste voorbeeld over het afval. Volgende opdracht is veel duidelijker voor de computer:

```
Ga tien stappen vooruit.
Open de deur.
Open jouw hand.
Beweeg jouw hand naar de blauwe zak.
Sluit jouw hand.
Hef jouw arm op.
…
```

De computer voert de stappen exact uit zoals wij die opstellen in ons **algoritme**. Een stappenplan dus om tot de juiste oplossing te komen.

Definitie:

> Een stappenplan dat een computer kan uitvoeren om tot de juiste oplossing te komen.

Wat ken je al? Voorkennis activeren!

Uit computationeel denken ken je al vijf belangrijke denkstappen.

Stap 1: Decompositie

Een groot probleem opdelen in kleinere stukjes.

Bijvoorbeeld: een doolhof oplossen is moeilijk als je naar de hele route tegelijk kijkt. Het wordt makkelijker als je de route opdeelt:

- eerst tot aan de bocht;
- dan draaien;
- dan verder tot aan het volgende kruispunt;
- dan opnieuw kiezen.

Stap 2: Patroonherkenning

Zoeken naar dingen die terugkomen of gekende programmeerconcepten herkennen.

Bijvoorbeeld: als je telkens dezelfde beweging moet uitvoeren, hoef je die niet telkens opnieuw apart te programmeren. Je kunt die stap laten **herhalen**. Een **herhaallus** is zo een gekend programmeerconcept.

Stap 3: Abstractie

Alleen letten op wat belangrijk is voor je oplossing. Bij een doolhof moet je niet letten op de kleur van de muren of op de vorm van het mannetje. Je moet vooral weten:

- waar start ik?
- waar moet ik eindigen?
- waar zijn de muren en bochten?
- welke stappen kan ik zetten?

Stap 4: Algoritme ontwerpen

Een algoritme is …

Definitie:

> `Een stappenplan dat een computer kan uitvoeren om tot de juiste oplossing te komen.`

Een algoritme moet precies genoeg zijn. Iemand anders, of een computer, moet het kunnen volgen, uitvoeren en dit kunnen zonder te raden wat jij bedoelt.

Stap 5: Debuggen

Definitie:

> `Fouten zoeken en die uit ons algoritme halen.`

De term '**debuggen**' klinkt vreemd, er zitten namelijk helemaal geen insecten in onze computers, maar dit komt door:

_____

_____

Debuggen doe je als volgt:

- Ik voorspel wat er zal gebeuren.
- Ik test mijn oplossing.
- Ik vergelijk mijn resultaat met mijn voorspelling.
- Ik zoek de eerste fout.
- Ik pas één ding aan.
- Ik test opnieuw.

Debuggen is geen extra stap enkel voor wie fouten maakt. Debuggen hoort bij programmeren.

Hoe werken we in deze cursus?

Je werkt met het online leerplatform Blockly in de Klas. Daarin zijn er vier oefensporen, telkens per twee verdeeld.

Doolhof / maze	Schildpad / Turtle
Blokken	**Blokken**
JavaScript-code	**JavaScript-code**

We starten telkens met het ontwikkelen van ons algoritme door gebruik te maken van blokken. Daarna vertalen we deze blokken naar JavaScript-code.

Sterke algoritmes

In deze cursus telt niet alleen of je programma werkt. Jouw oplossingen moeten '**sterk**' zijn.

Een oplossing is sterker wanneer je ook kunt uitleggen:

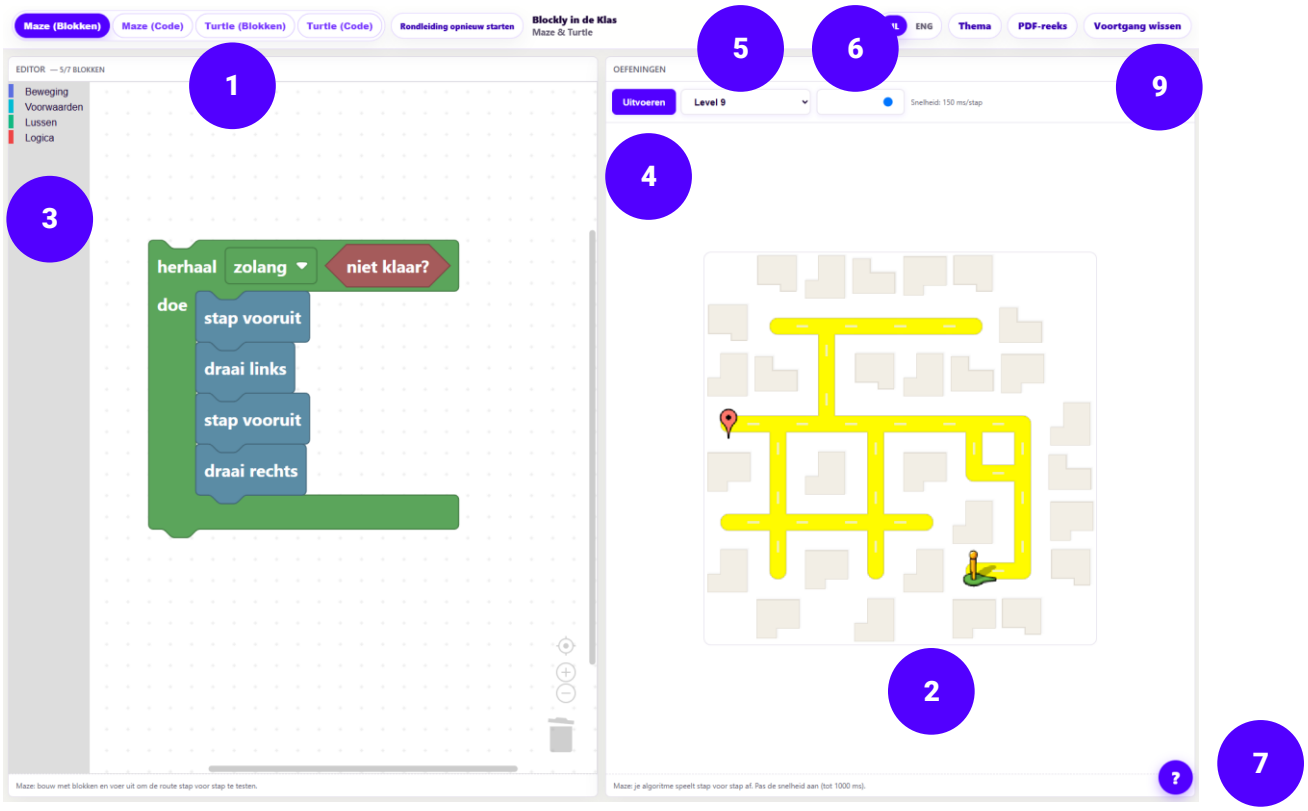
- welk probleem je moest oplossen;
- welke stappen je hebt gebruikt;
- waar je een patroon zag;
- waar je programma moest kiezen;
- welk stuk herhaald werd;
- welke fout je vond;
- hoe je die fout hebt verbeterd.

Een programma dat toevallig werkt, is nog geen sterke oplossing.

Een **sterk algoritme** werkt, is duidelijk opgebouwd en kan door iemand anders begrepen worden. Het is met andere woorden **functioneel** én **leesbaar**.

Rondleiding

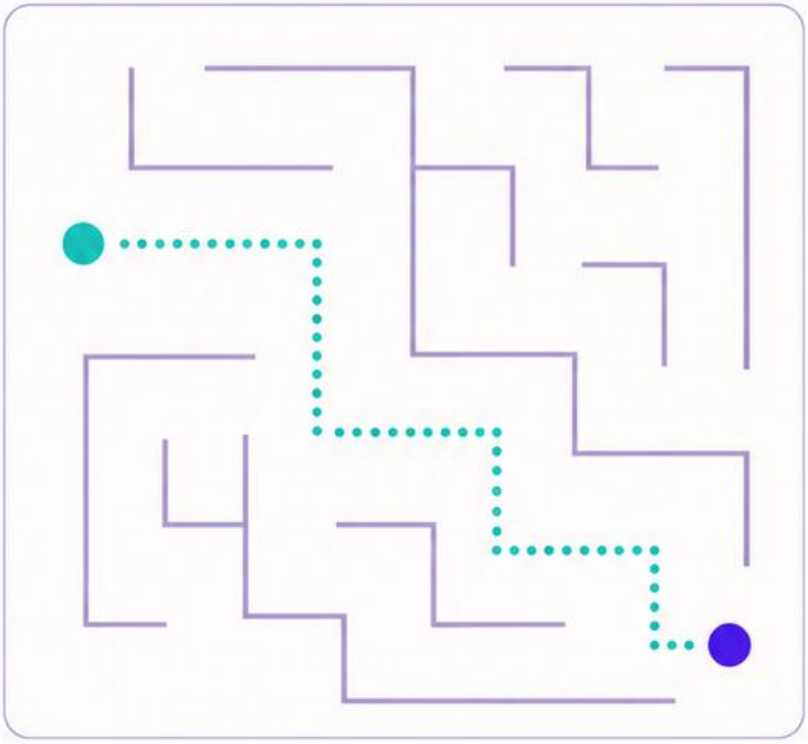
We werken in deze cursus met **Blockly in de Klas**. Dat is een online leerplatform waarin je leert programmeren met blokken en JavaScript-code. Je hoeft niets te installeren. Je opent de website in de browser en kunt meteen starten.



1) **De vier oefenmodi:** hier kies je welke oefeningen je wilt maken.
2) **De oefening**: hier zie je het probleem dat jij moet ontleden en oplossen met een algoritme.
3) **De editor**: in de editor bouw je jouw algoritme met blokken of met JavaScript-code.
4) **De knop uitvoeren**: is jouw algoritme klaar en wil je debuggen, dan klik je op deze knop.
5) **Oefeningen**: via dit drop-down menu kan je een ander level openen.
6) **Snelheidsbalk**: hier pas je de snelheid aan van de uitvoer van jouw algoritme.
7) **Het formularium**: zit je heel even vast bij de syntax, dan kan je hier even spieken. Opgelet: gebruik dit enkel als jij zelf al een poging ondernam om het algoritme te ontwikkelen. Door meteen te spieken daag je jouw hersenen niet uit en treedt er geen leereffect op. Daarom altijd eerst zelf proberen en leren uit jouw fouten.
8) **Feedback**: wanneer je een oplossing uitvoert, geeft het platform meteen feedback. Lees die grondig na bij het debuggen of om jouw algoritme te verbeteren.
9) **Vooruitgang**: het platform bewaart jouw vooruitgang in de browser. Je kan altijd jouw vooruitgang wissen als je opnieuw wil starten. Opgelet: werk je op een andere computer, dan zal jouw vooruitgang daar niet op staan.
10) **PDF-export**: hiermee kan je jouw oefeningen exporteren en indienen bij de vakdocent.

Maze-oefeningen

- **computationeel** denken
- **routes** plannen
- **sequenties** gebruiken
- **selectie** gebruiken
- **herhalen** zolang niet klaar



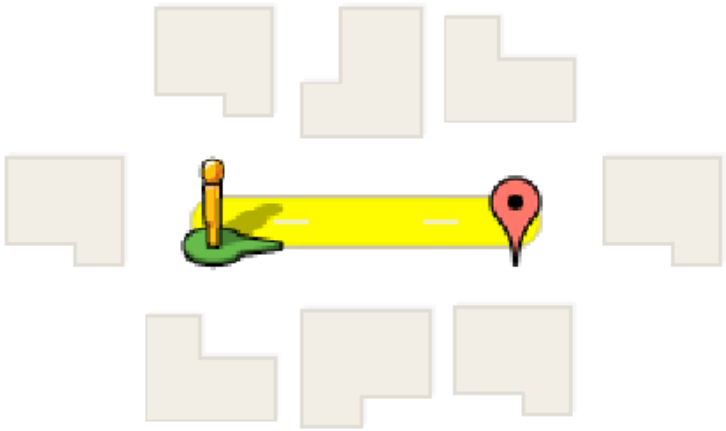

```
1  while (notDone()) {
2    moveForward();
3  }
```

Hoofdstuk 1: doolhof

Level 1: Maze met blokken

In dit hoofdstuk gaan we aan de slag met de eerste opdrachten in Maze. We sturen een mannetje door een doolhof en leren hoe we een route kunnen omzetten naar een algoritme. We beginnen met blokken. Blokken helpen je om eerst na te denken over de volgorde van je stappen. Je hoeft nog niet meteen bezig te zijn met hoofdletters, haakjes of puntkomma's. Daarna maken we de vertaalslag naar JavaScript-code.

Het programmeerconcept dat centraal staat in dit hoofdstuk is de sequentie. Bij een sequentie worden instructies één voor één uitgevoerd. De volgorde is belangrijk. Als je twee stappen omwisselt, kan het hele programma fout lopen.



1.1 Het doolhof lezen (decompositie)

Bij een doolhof let je op vijf dingen:

- Waar start het mannetje?
- In welke richting kijkt het mannetje?
- Waar ligt het doel?
- Waar staan de muren?
- Welke stappen zijn nodig om bij het doel te raken?

Dat is **decompositie**. Je deelt het probleem op in kleinere stukjes.

Je denkt dus niet: "Ik moet het hele doolhof oplossen", maar wel.

- Daarna nog eens vooruit.
- Dan moet ik draaien.
- Daarna moet ik opnieuw vooruit.

Zo wordt het probleem kleiner en kleinere problemen zijn makkelijker op te lossen.

1.2 Algoritme ontwerpen - schematisch (blokken)

Maak hieronder een korte schematische voorstelling van het doolhof en van jouw algoritme met blokken. Daarna testen we onze oplossing en gaan we op zoek naar bugs.

1.3 Syntax ontleden (code)

Nu bekijken we dezelfde oplossing of algoritme, maar uitgeschreven in JavaScript-code. Een blok zoals 'stap vooruit' wordt dan:

```
moveForward();
```

Bekijk de coderegel heel goed. Vergelijk met hoe je dit in het Engels zou schrijven.
Welke verschillen vallen jou op? Geef er minstens vier.

Je hoeft nu nog niet alles over JavaScript te kennen. Je moet vooral leren dat code zeer precies geschreven moet worden. Deze twee regels zijn **niet hetzelfde:**

```
moveForward();
```

moveforward();

De tweede regel is fout omdat de hoofdletter ontbreekt. De JavaScript-compiler is heel streng. De computer raadt dus niet wat je bedoelt. Code is precies correct of fout.

1.4 Syntax ontleden (GRM)

Neem jouw grafisch rekentoestel er even bij en tik volgende bewerking in:

5*5+

Druk daarna op Enter. Welke foutmelding verschijnt er op het scherm? Kan je die foutmelding ook duiden met wat je hier hebt geleerd?

1.5 Syntax ontleden (online)

Ga naar de codevariant van level 1. Tik daar onderstaande foutieve code in.
Welk resultaat krijg je van de webtool?

Voer in:

moveforward();

moveforward()

Welke fouten vind je in de code hierboven? Debug!

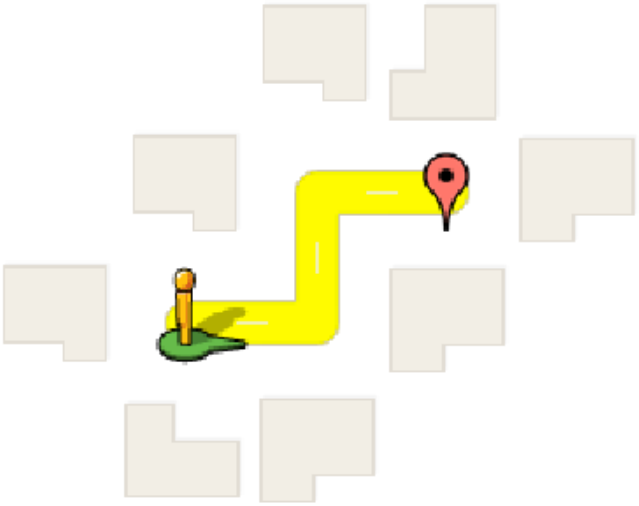
Welke uitvoer krijg je van het webplatform na het invoeren van deze foute code?

Welke opmerking in de uitvoer krijg je als je enkel de puntkomma's vergeet?

Level 2: zelf aan de slag met de sequentie

In oefening 1 hebben we samen bekeken hoe je een route in een doolhof omzet naar blokken en daarna naar JavaScript-code. Level 2 lijkt sterk op level 1. Er komt geen nieuw programmeerconcept bij. Je werkt opnieuw met **sequentie**: stappen die één voor één in de juiste volgorde worden uitgevoerd.

Het verschil is dat je nu zelfstandiger werkt. Je gaat eerst het doolhof lezen, dan een schema maken met blokken, daarna je code op papier schrijven en pas op het einde testen in de browser.



Stap 1: decompositie

Bekijk bovenstaande opgave.
Welke nieuwe acties hebben we nodig om tot het einde van ons doolhof te raken?

Stap 2: ontwerp jouw algoritme met blokken

Teken hieronder een schema van jouw oplossing. Gebruik hiervoor blokken.

Stap 3: van Nederlands naar Engels naar JavaScript

De blokken in het platform gebruiken Nederlandse woorden, maar JavaScript gebruikt Engelse commando's. Vul de vertaling aan:

Nederlands	Engels	JavaScript
`while`		
`notDone()`		
`moveForward();`		
`{ … }`		

Stap 4: ontwerp op papier

Schrijf nu de JavaScript-code voor jouw oplossing.

Controleer je code voor je ze test.

- **Elk commando staat op een aparte regel.**
- **Elk commando heeft haakjes: ()**
- **Elke actie eindigt met een puntkomma: ;**
- **De hoofdletters staan juist.**
- **left en right zijn niet omgewisseld.**
- **De volgorde klopt met mijn schema.**

Stap 5: test en debug jouw algoritme

Tik jouw code over van je papier.

> Let op: kopieer niet zomaar van iemand anders. Je test jouw eigen voorspelling.

Voer de code uit. Wat gebeurde er?

- Mijn code werkte meteen.
- Mijn code werkte niet meteen.
- Het mannetje botste tegen een muur.
- Het mannetje draaide verkeerd.
- De browser gaf een foutmelding.
- Iets anders: _____

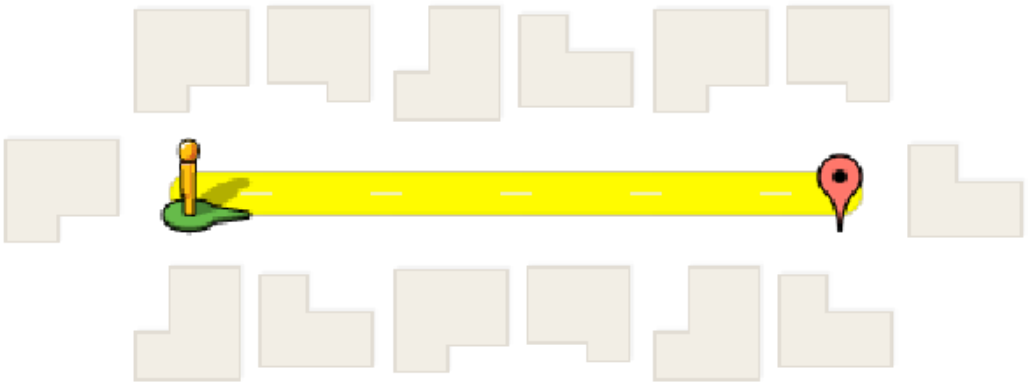
Was je voorspelling juist?

ja / nee / gedeeltelijk

Leg uit:

Schrijf jouw verbeterde code. Duid met kleur aan wat je hebt aangepast.

Level 3: patroonherkenning en herhaling



Bij oefening 1 en 2 werkte je vooral met **sequentie**. Je zette stappen gewoon na elkaar:

```
vooruit
vooruit
rechts draaien
vooruit
```

Dat werkt prima voor korte routes. Maar wat als een route langer wordt? Dan krijg je al snel een programma dat er zo uitziet:

```
vooruit
vooruit
vooruit
vooruit
vooruit
vooruit
vooruit
vooruit
```

Dit is weinig efficiënt voor jou als mens om uit te schrijven. Je schrijft telkens opnieuw hetzelfde. Wanneer je vaak hetzelfde ziet terugkomen in een algoritme, moet er een belletje rinkelen.

Hier komen we het programmeerconcept **patroonherkenning** (opnieuw) tegen.

Je zoekt naar een patroon in de code dat vaak terugkomt of een patroon dat je herkent. Dat probeer je sneller, korter en beter te schrijven in jouw algoritme.

Niet:

Ga vooruit. Ga vooruit. Ga vooruit. Ga vooruit. Ga vooruit. Ga vooruit. Ga vooruit.

Maar wel:

Blijf vooruitgaan **zolang** je nog niet klaar bent.

Dat is korter en het toont beter wat je algoritme eigenlijk bedoelt.

Stap 1: Patroonherkenning

Open het leerplatform en open level 3. Bekijk het doolhof zonder een algoritme te ontwerpen. Beantwoord volgende vragen:

Welke stap komt de gehele tijd terug?

_____

Wanneer stoppen we met ons algoritme?

_____

Dus, je blijft herhalen ZOLANG:

_____

_____

Stap 2: algoritme ontwikkelen (schematisch)

Maak nu jouw oplossing door gebruik te maken van de blokken. Teken hieronder jouw oplossing:

Stap 3: algoritme ontleden (codetaal / syntax)

We bekijken hetzelfde algoritme in codetaal en ontleden deze om de **SYNTAX** te begrijpen.

Wat betekent syntax ook weer bij het programmeren?

_____

_____

Onze code ziet er als volgt uit:

```
while (notDone()) {
moveForward();
}
```

We ontleden de syntax aan de hand van onderstaande tabel:

Element	In het Nederlands	Opletten voor …	Type
while			
notDone()			
moveForward();			
{ … }			

Stap 4: Even herhalen …

Deze oefening ging vooral over patroonherkenning. Je ziet hieronder twee algoritmes.

Welke algoritmes zijn juist?

_____

_____

Welk algoritme is beter? Leg ook uit waarom en voor wie.

_____

_____

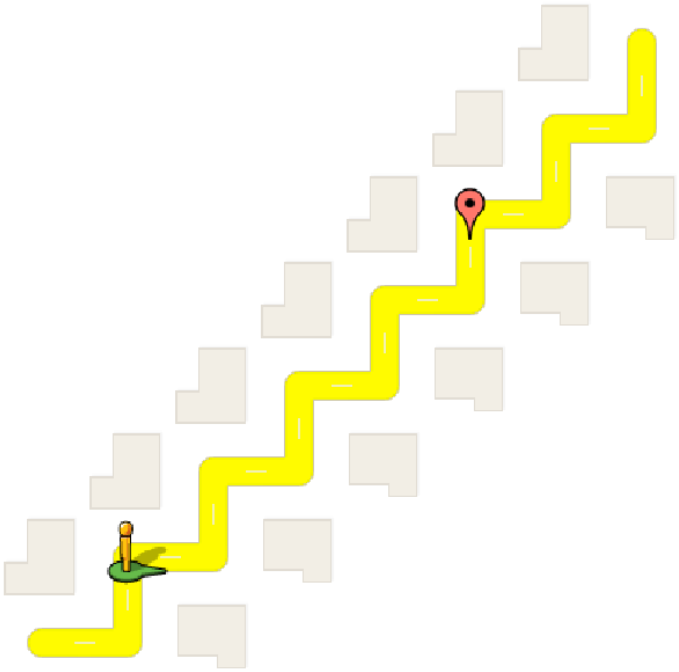

```
while (notDone()) {
moveForward();
}
```

Level 4: combineren!

In oefening 4 krijg je geen volledig nieuw programmeerconcept. Je herkent twee dingen die je al eerder hebt gezien:

1. Uit level 2 herken je een route met:
    - vooruit;
    - links draaien;
    - rechts draaien.
2. Uit level 3 herken je:
    - herhalen **zolang** het doel nog niet bereikt is.

Deze oefening gaat dus over combineren. Je gebruikt opnieuw **patroonherkenning** om de acties en de herhaallus op te sporen.



Stap 1: voorspellen

Tijdens deze stap gaan we eerst trachten om ons algoritme in schematische voorstelling te voorspellen. Daarna vertalen we ons schema naar codetaal. Tenslotte zullen we ons schema en de code controleren en debuggen.

Bekijk de route. Maak een voorspelling en vul hieronder jouw schema's in. Denk aan het programmeerconcept uit level 3: herhaal zolang niet klaar …

Blokken	Codetaal

Stap 2: testen en debuggen

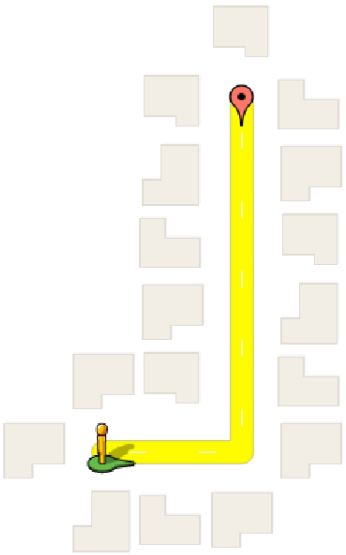
Open de oefenomgeving en test jouw algoritmes.
Werk eerst met blokken en daarna met de JavaScript-code.

Duid aan:

- Mijn blokkencode werkte goed / niet goed.
- Ik vond volgende fout in mijn blokkencode: _____
- Mijn JavaScript-code werkte goed / niet goed.
- Ik vond een fout bij:
    - Ik maakte een hoofdletterfout bij ‘while’.
    - notDone() stond niet tussen ronde haakjes.
    - Ik maakte een hoofdletterfout bij notDone().
    - Ik maakte een hoofdletterfout bij een van de acties (moveForward enz.).
    - Ik had links en rechts omgewisseld.
    - Er stond een stap buiten de accolades.
    - Ik vergat de accolades.
    - Anders: _____

Level 5: binnen en buiten de lus

In level 5 leer je geen nieuw programmeerconcept, nieuwe syntax of een nieuwe actie. In dit level oefenen we onze vaardigheid bij de **decompositie** en **patroonherkenning**. Wat hoort wél in de herhaallus en wat hoort niét in de herhaallus?



Stap 1: decompositie

Bekijk het doolhof.
We delen onze route op in twee delen.

Stappen die maar één keer gebeuren (noteer in JS-code, let op voor de syntax):

Stappen die meerdere keren moeten gebeuren (noteer in JS-code, let op voor de syntax):

Stap 2: algoritmes ontwikkelen

Teken eerst in de linkerhelft jouw blokkenschema.
Vertaal daarna jouw schema naar codetaal. Let op voor de correcte syntax!

Blokken	Codetaal

Stap 3: testen en debuggen

Open de oefenomgeving en test jouw algoritmes. Eerst in blokken en daarna met de JavaScript-code.

Duid aan:

- Mijn blokkencode werkte goed / niet goed.
- Ik vond volgende fout in mijn blokkencode: _____
- Mijn JavaScript-code werkte goed / niet goed.
- Ik vond een fout bij:
  - Ik maakte een hoofdletterfout bij ‘while’.
  - notDone() stond niet tussen ronde haakjes.
  - Ik maakte een hoofdletterfout bij notDone().
  - Ik maakte een hoofdletterfout bij een van de acties (moveForward enz.).
  - Ik had links en rechts omgewisseld.
  - Er stond een stap buiten de accolades.
  - Ik vergat de accolades.
  - Anders: _____

Stap 4: leesbare code schrijven

Bekijk de code in de oefenomgeving en hieronder grondig. Welk verschil valt jou op?

Code A	Code B

`while (notDone()) {`
`  moveForward();`
`turnLeft();`
`}`
	`while (notDone()) {`
	`  moveForward();`
	`  turnLeft();`
	`}`

Het verschil: _____

_____

Dit heet een: _____

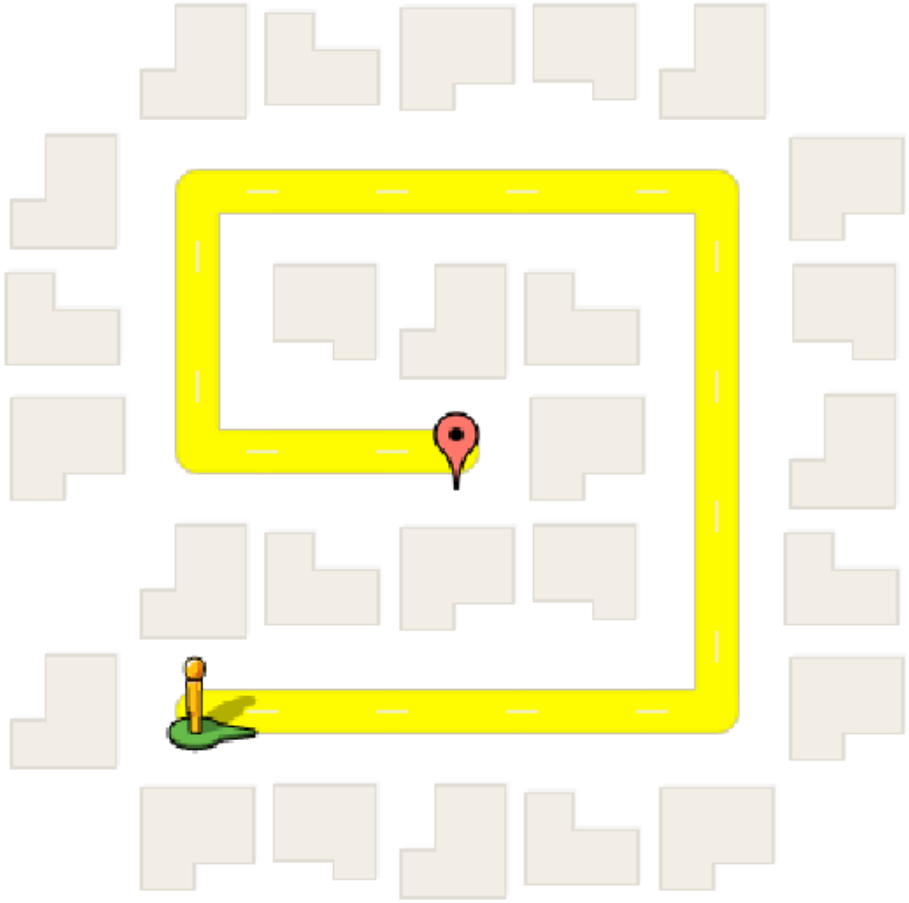
Code B is beter dan Code A, want:

_____

_____

Een **inspringing** of een **indent** is dus geen versiering. Het helpt je om de structuur van je algoritme te zien. Bij evaluaties, practica en examens wordt er ook gelet op het schrijven van leesbare code!

Level 6: kiezen met ‘if’



Tot nu toe bouwden we onze routes in het doolhof met:

- Stappen in volgorde (=**de sequentie**)
- Herhaling met while
- Herkenbare patronen

In level 6 komt er een nieuw programmeerconcept bij: **selectie**.
Bij de **selectie** laat je het programma een keuze maken. Bijvoorbeeld:

“Als er een pad is naar links, dan draai je naar links.”

Stap 1: actie bepalen

Bekijk onderstaand doolhof. Duid met twee kleuren de zones waarin de computer een actie moet uitvoeren. Welke twee acties komen terug in ons algoritme?

Kleur 1: _____ Kleur 2: _____

Stap 2: voorwaarde bepalen

Beide acties worden uitgevoerd wanneer aan een voorwaarde is voldaan. Dat wil zeggen dat de computer zichzelf een vraag stelt. Als het antwoord daarop 'Waar' of 'True' is, voert het de actie uit.

Wat zijn de voorwaarden bij beide acties?

Actie 1: _____ **Actie 2:** _____

Stap 3: algoritme ontwikkelen

Teken eerst in de linkerhelft jouw blokkenschema en test jouw algoritme uit in de oefenomgeving. Vertaal daarna jouw schema naar codetaal. Let op voor de correcte syntax!

Blokken	Codetaal

Stap 4: tweemaal inspringen

Bekijk de code grondig. Er valt ons op dat de code twee delen kent die 'inspringen'.

- **Duid in stap 3 deze twee delen aan in twee verschillende kleuren.**
- **Waarom werken we hier met twee soorten 'indents' of 'insprongen'?**

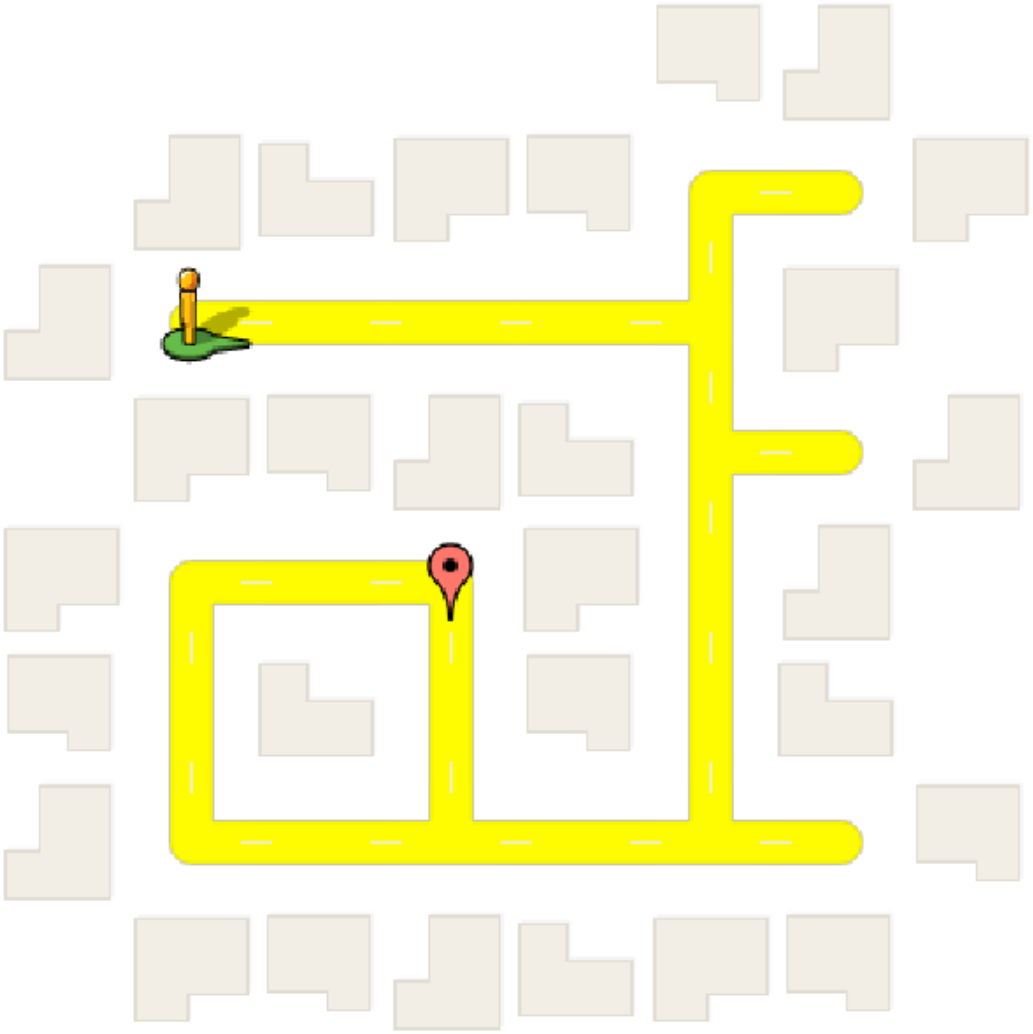
_____

_____

_____

_____

Level 7: opgelet voor de oneindige lus



Level 7 lijkt heel sterk op level 6. In level 6 leerde je werken met de **selectie**. Daarbij controleert de computer een **voorwaarde**:

"Is er een pad links?"

Als dat **waar** of '**True**' is, voer het de actie uit:

"Draai naar links."

In level 7 gebeurt nagenoeg hetzelfde, maar we moeten hier letten op twee zaken:

- De richting bij de voorwaarde
- De richting bij de actie
- De oneindige lus

We staan even stil bij dat laatste en beantwoorden volgende vragen:

- Wat is een oneindige lus?
- Waarom vormt dit een probleem?
- Hoe vermijd ik oneindige lussen?

Stap 1: Wat is een oneindige lus

Kopieer de code uit level 6 en test deze uit bij level 7. Doe dit eerst door zelf een voorspelling te maken. Wat zal er gebeuren bij het doolhof uit level 7 wanneer de computer het algoritme uit level 6 uitvoert?

Waarom zal de computer dat zo uitvoeren?

Een **oneindige lus** ontstaat wanneer een herhaling blijft doorgaan en nooit stopt.

Bijvoorbeeld:

```
while (notDone()) {
  moveForward();
}
```

Deze code betekent:

"Blijf vooruitgaan zolang het doel nog niet bereikt is."

Dat is alleen oké als het mannetje ooit het doel kan bereiken.

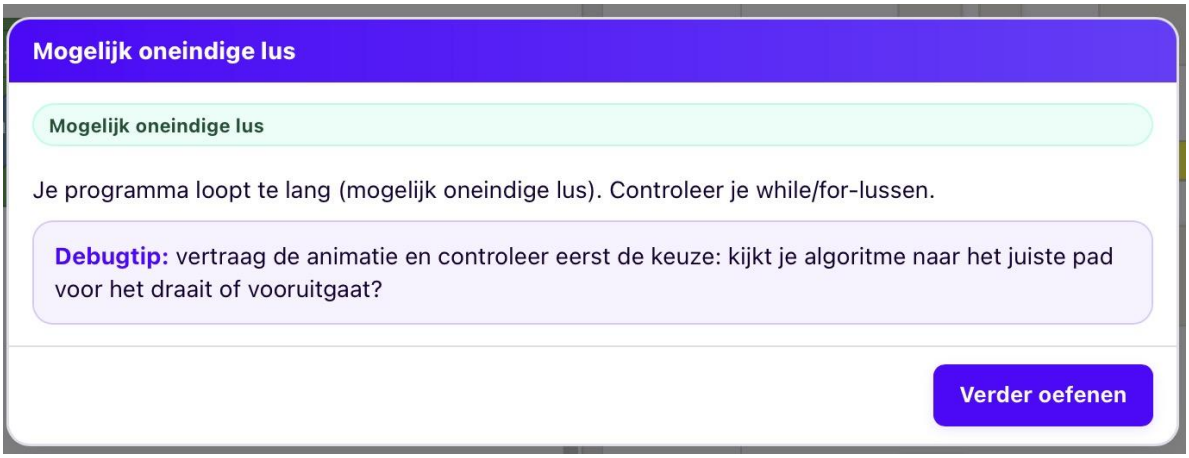
```
Als het mannetje vastloopt tegen een muur en het doel niet bereikt, blijft notDone()
waar. De while blijft dan opnieuw proberen.
```

De computer denkt niet:

"Dit lukt precies niet. Ik stop maar. Ik ben het beu."

De computer blijft uitvoeren wat jij hebt geschreven.

> **Opgelet**: een oneindige lus is géén syntaxfout, maar een fout in ons algoritme. De computer doet exact wat wij vroegen … maar helaas was dat niet de oplossing voor ons probleem. Een oneindige lus kan ervoor zorgen dat een programma trager loopt of zelfs crasht.



Stap 2: oneindige loop in mijn game

Lees de tekst hieronder en beantwoord de bijhorende vragen.

> In oefening 7 zagen we dat een while-lus gevaarlijk kan worden wanneer het programma blijft herhalen zonder het doel te bereiken. Dat soort probleem bestaat niet alleen in kleine oefeningen. Ook grote games kunnen fouten bevatten waarbij een computer te vaak hetzelfde controleert.
>
> Bij de pc-versie van *Monster Hunter Wilds* werd een prestatieprobleem onderzocht. In bepaalde gebieden van het spel controleerde het systeem veel te vaak of spelers bepaalde extra **DLC** hadden gekocht. Die controle gebeurde vooral in de buurt van de Support Desk, een plaats waar spelers in het spel extra inhoud kunnen bekijken.
>
> Daardoor moest de computer voortdurend extra werk doen. Dat zorgde niet voor een foutmelding zoals bij een ontbrekend haakje of een verkeerd gespeld commando. Het spel bleef wel werken, maar de prestaties konden heel sterk dalen. Spelers merkten dat aan **FPS**, waardoor het spel minder vlot aanvoelde.

In het tekstfragment komen twee acroniemen (letterwoorden) aan bod. Wat betekenen deze?

DLC: _____

Voorbeeld: _____

FPS: _____

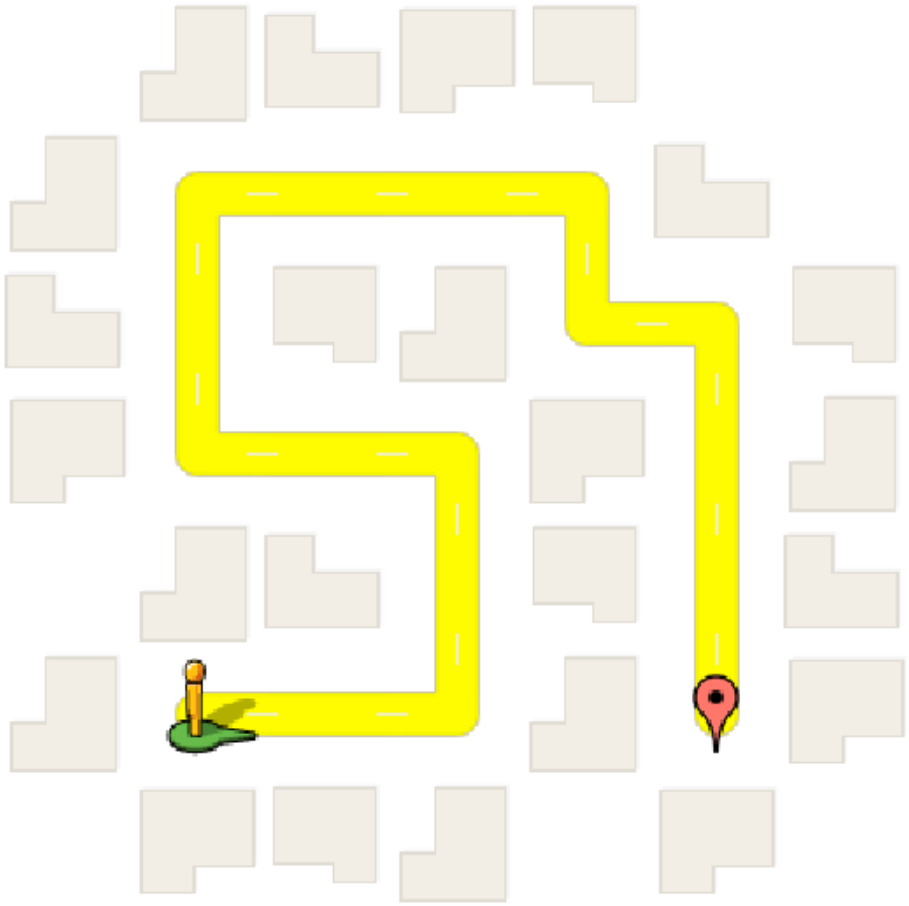
Voorbeeld: _____

Wat is de link tussen het artikel over ‘Monster Hunter’ en ons Blockly-level?

_____

_____

Level 8: links, rechts, links, rechts …



In dit level leer je geen nieuw programmeerconcept, maar proberen we ons algoritme uit het hoofd te schrijven. Dit doen we stap voor stap.

Stap 1: even herhalen …

Bekijk onderstaande tabel en vul aan:

Actie	Actie in JS	Wanneer?	Voorwaarde in JS
Naar voor bewegen			
Naar links draaien			
Naar rechts draaien			

Stap 2: code uit het hoofd

Noteer hieronder jouw algoritme in JavaScript-code:

Stap 3: Debuggen

Controleer jouw code, ga op zoek naar de fouten en haal deze uit jouw algoritme. Vul daarna onderstaande checklist aan.

- Ik koos de verkeerde voorwaarde.
- Ik koppelde de verkeerde actie aan de voorwaarde.
- Ik wisselde links en rechts om.
- `moveForward();` stond op de verkeerde plaats.
- Een commando stond buiten de `while`, maar moest erin.
- Een commando stond binnen een `if`, maar moest daar niet staan.
- Ik maakte een syntaxfout.
- Mijn code was juist.

Wat heb je aangepast?

Level 9: als het niet waar is: else

In de voorgaande levels leerde je reeds herhalen met een while-statement en keuzes maken met een if-statement. Bij dat laatste bepalen wij een voorwaarde. Als die voorwaarde 'waar' of 'true' is, dan voeren we een actie uit. Maar wat als niet aan die voorwaarde wordt voldaan? Dus 'false'? Dan gebeurt er helemaal niks. Skip! … Tenzij we gebruikmaken van een else statement! Anders kan namelijk ook!

Stap 1: else met blokken

Analyseer het algoritme en vul onderstaande vragen in.

Welke twee voorwaarden zitten in dit algoritme?

Wat gebeurt er als de eerste voorwaarde true of waar is?

Wat gebeurt er als de tweede voorwaarde 'true' of waar is?

Wat gebeurt er als de tweede voorwaarde false of onwaar is?

Stap 2: else met code (syntax)

Deze keer gieten we de blokken in codetaal en analyseren we het verschil tussen het if-statement en het else-statement.

Blokken	Codetaal

	`while(notDone()) {`
	`  if(isPathAhead()) {`
	`  moveForward();`
	`  }`
	`  else {`
	`  turnLeft();`
	`  }`
	`}`

Welke voorwaarde gebruiken we om na te gaan of er een pad voorwaarts is?

Wat heeft een if-statement dat een else-statement niet heeft?

In deze code staan verschillende accolades. Welke functie hebben deze accolades?

In deze code staan verschillende inspringingen. Welke functie hebben deze ‘indents’?

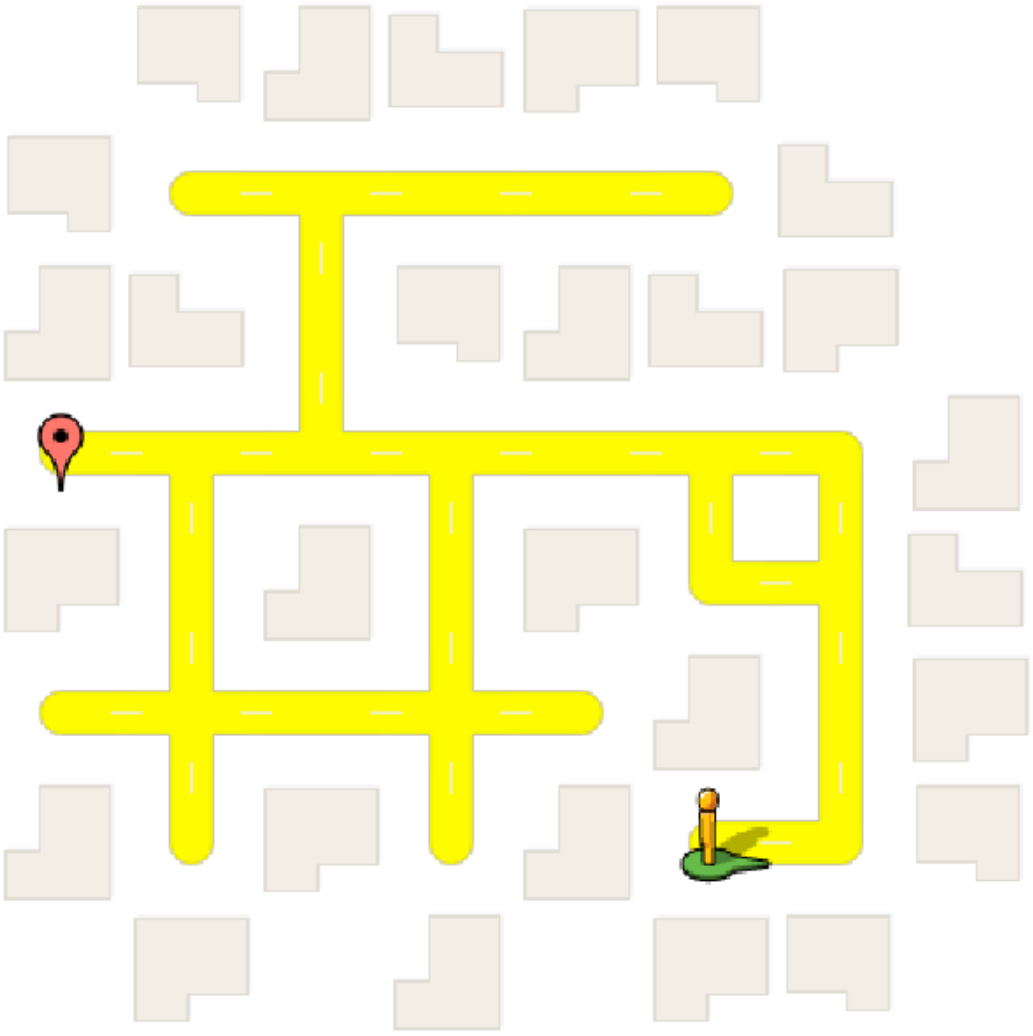
Stap 3: Aan de slag!

Bekijk volgende doolhof in level 9 op het webplatform. Beantwoord volgende vragen:

Wat moet het mannetje doen als er een pad vooruit is?

Wat moet het mannetje doen als er geen pad vooruit is?

Bouw jouw algoritme door gebruik te maken van een else-statement. Als iets niet waar is, gebeurt X. Start eerst met blokken. Daag daarna jezelf uit door level 9 in code te maken!



Stap 4: Syntax herhalen

Een if zegt wat er gebeurt wanneer een voorwaarde **waar** is.

Een else zegt wat er gebeurt wanneer diezelfde voorwaarde **niet waar** is.

```
if (isPathAhead()) {
  moveForward();
} else {
  turnLeft();
}
```

Dit betekent:

Als er een pad vooruit is, ga vooruit.
Anders, draai links.

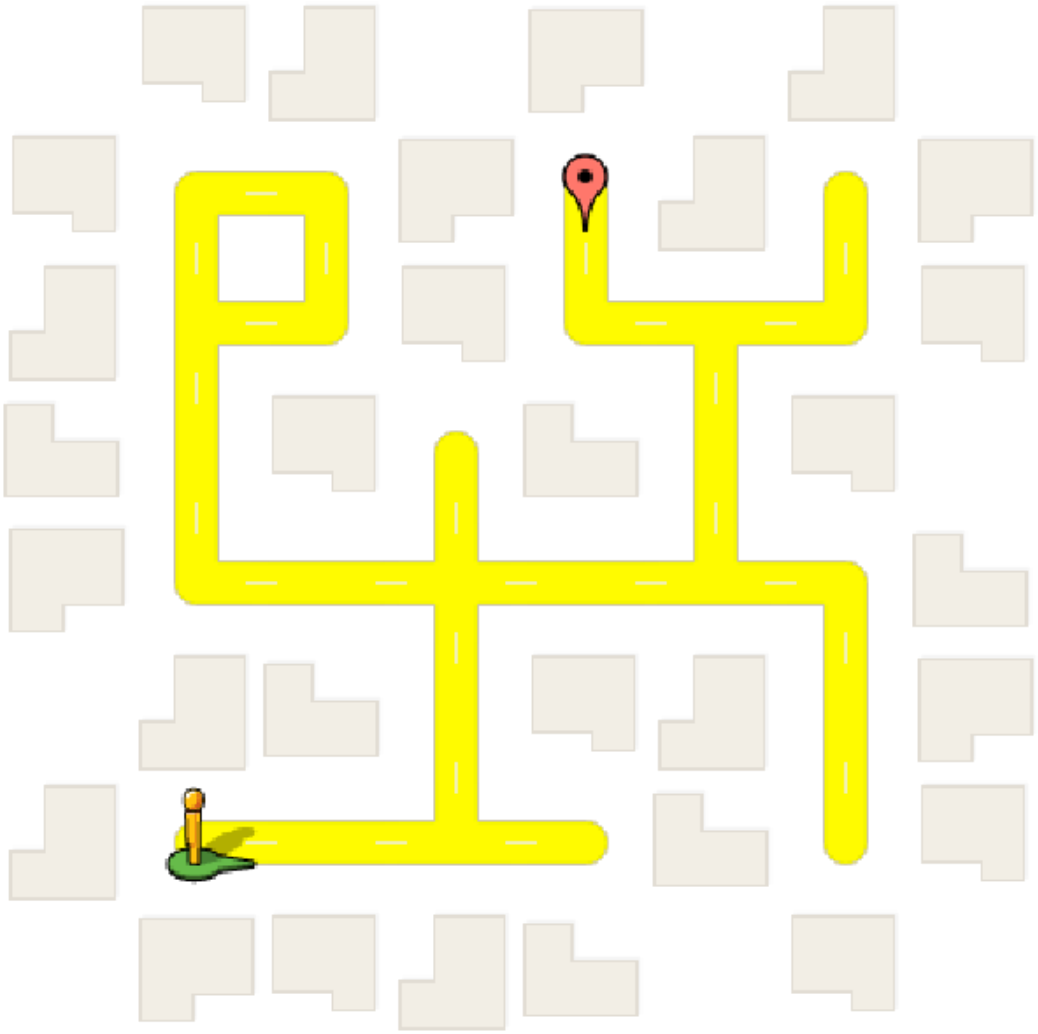
Opgelet: Een else-statement is dus geen tweede losstaand element. De else hoort bij de if.

Level 10: eindbaas

Level 10 is het einddoolhof van dit hoofdstuk. Deze oefening is moeilijker dan de vorige oefeningen, maar er komt eigenlijk geen enkel nieuw programmeerconcept aan te pas. Je kan dit algoritme bouwen met alle stukken die je reeds hebt geleerd:

- while om te blijven herhalen zolang het doel nog niet bereikt is;
- if om een keuze te maken;
- else om te zeggen wat er gebeurt als de voorwaarde niet waar is;
- moveForward(); om vooruit te gaan;
- turnLeft(); om links te draaien;
- turnRight(); om rechts te draaien.

> De opdracht lijkt initieel moeilijk omdat alles samenkomt. Daarom krijg je volgende tip: **je hebt elk geleerd blokje code één en slechts éénmaal nodig om het einde te bereiken.** Veel succes!



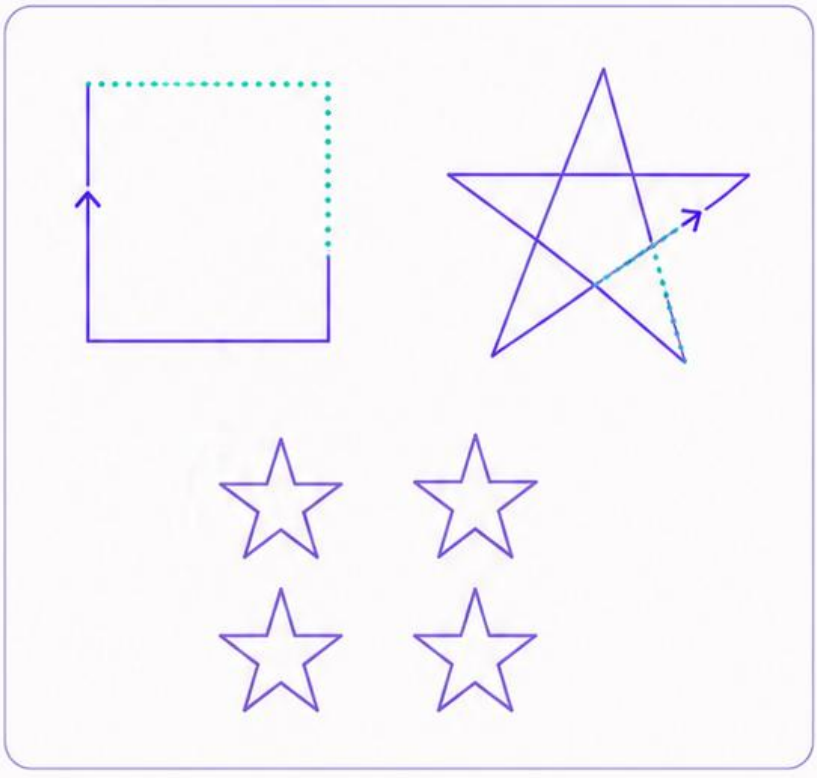
Schildpad-oefeningen

patronen tekenen

draaihoeken gebruiken

begrensde herhalingen gebruiken

parameters instellen




```
for (var count = 0; count < 4; count++) {
  moveForward(100);
  turnRight(90);
}
```

Hoofdstuk 2: Schildpad

Level 1: Vierkant tekenen

In het vorige hoofdstuk leerde je om met acties en voorwaarden een algoritme te bouwen om een mannetje doorheen een doolhof te sturen. In dit hoofdstuk gaan we aan de slag met '*turtle*' of schildpadoefeningen. In deze oefeningen beweeg je een pen om figuren op het scherm te tekenen.

De schildpad kan volgende bewegingen maken in de eerste levels:

- vooruit bewegen
- draaien overheen een draaihoek

Stap 1: Decompositie en patroonherkenning

Bekijk volgend schema dat we moeten tekenen en beantwoord de bijhorende vragen:

Hoe ver moet onze schildpad vooruit bewegen? Welke eenheid gebruik we daarvoor?

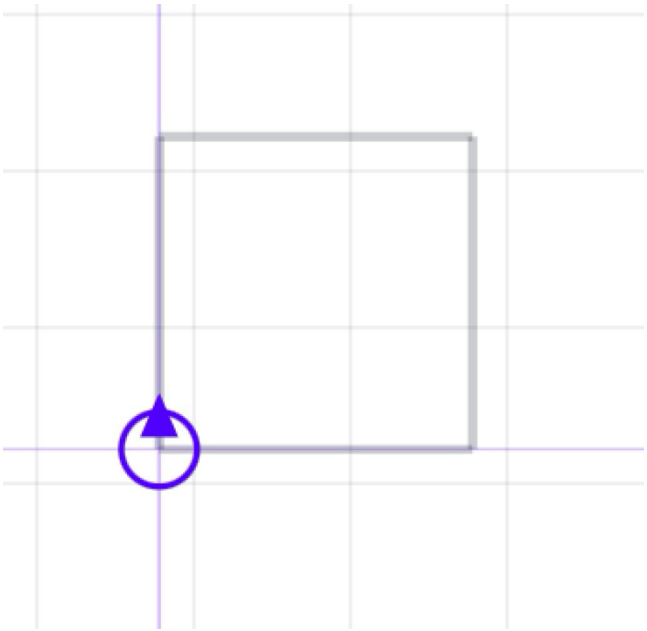
_____

_____

Hoeveel graden moet de schildpad draaien?

_____

Teken een draaihoek A op het schema.



Welk patronen herkennen we in dit schema? Schrap wat niet past:

sequentie – selectie – voorwaardelijke herhaling – begrensde herhaling

Hoeveel keer worden onze acties herhaald in het schema? _____

Stap 2: Parameters en eenheden ontleden

Uit het vorige hoofdstuk ken je de codes om onze sprite vooruit te latne bewegen of te laten draaien naar links of rechts. Deze komen hier terug, maar we vullen dit aan met **parameters**. Bekijk volgende codes:

```
moveForward(100);
turnRight(90);
```

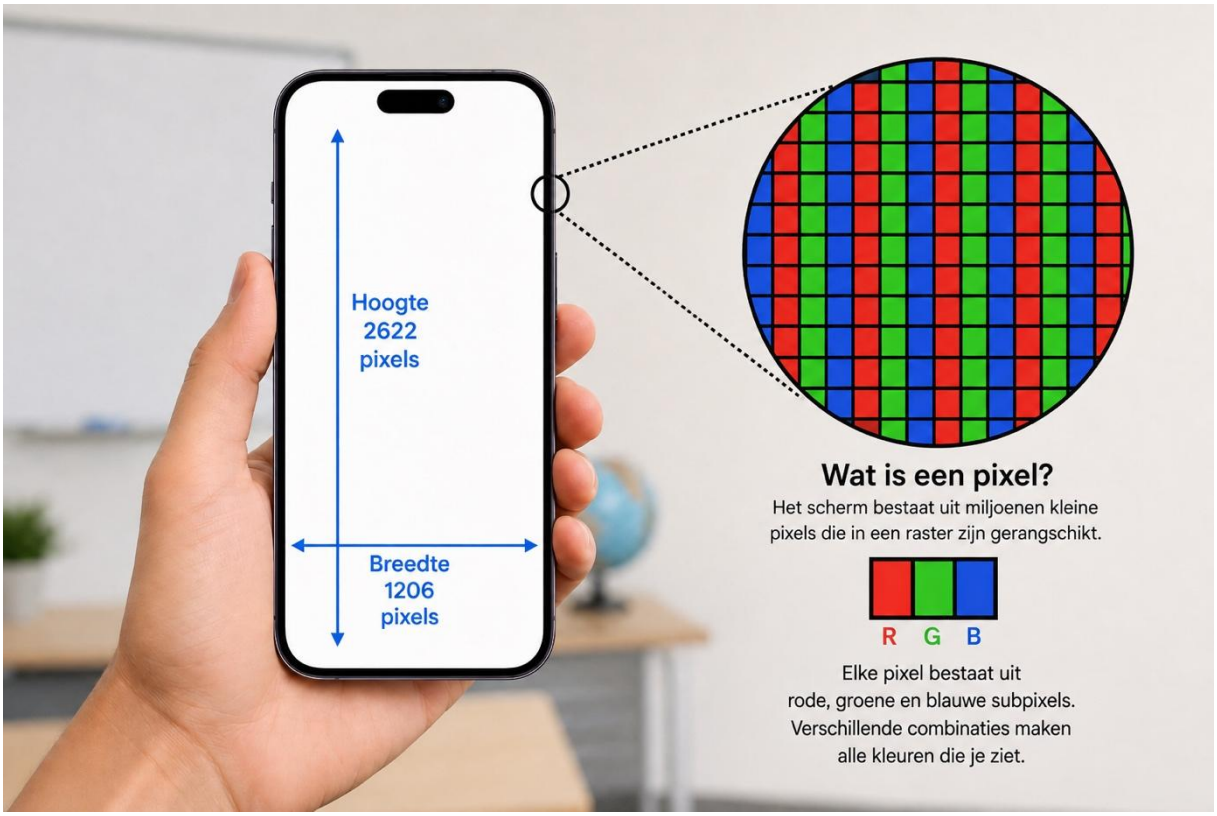
Een commando als **moveForward(100);** betekent:

- beweeg vooruit
- over een afstand van **100 pixels**.

Definitie:

> `Dat getal tussen de haakjes noemen we een parameter. Een parameter geeft extra informatie aan een commando of actie. Een pixel is de eenheid en slaat op de kleine vierkantjes / rechthoekjes die in jouw beeldscherm zitten.`

Stel dat je een televisiemonitor thuis hebt met een resolutie van 4K, dan betekent dit doorgaans dat de basis van het scherm 3840 kleine lichtgevende pixels telt en de hoogte er 2160 telt. Door de basis naar boven af te ronden komen we uit bij 4000 of 4K. Met een commando `moveForward(100);` springt onze schildpad dus 100 pixels verder op het scherm.



Bekijk volgend commando en beantwoord de bijhorende vragen:

```
turnRight(90);
```

Wat is in dit commando de parameter: _____

Wat bepaalt deze parameter: _____

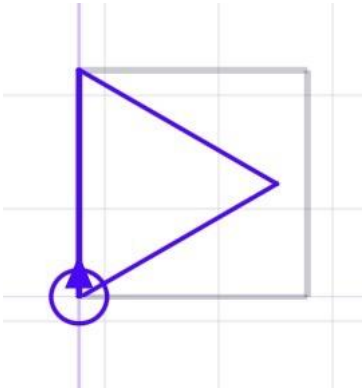
Wat is de eenheid van deze parameter: _____

Stap 3: Supplementaire hoeken

Bij ons vierkant zie je vier rechte hoeken. Een rechte hoek is altijd 90 graden. Daarom gebruiken we hier volgend commando:

```
turnRight(90);
```

Toch moeten we hier goed opletten, want de **hoek die je ziet** in de figuur **is niet altijd** dezelfde als de **hoek waarmee de schildpad draait (= de draaihoek).**



```
Zichtbare hoek in het Vierkant = 90 graden
Draaihoek van de schildpad = 90 graden
```

Bij een driehoek is dit anders. Daar zijn de hoeken binnen de figuur 60 graden en is de draaihoek van onze schildpad 120 graden. Let op dat de som van de hoek binnen de figuur en de **draaihoek** elkaars **supplement** zijn. **Supplementaire hoeken** vullen elkaar aan. De som van beide hoeken bedraagt altijd 180 graden.

Stap 4: Vierkant zonder herhaling

Hieronder zie je een algoritme om het vierkant te tekenen.
Het algoritme werkt, maar is niet optimaal.

Bekijk de code en duid het stuk code aan dat telkens terugkomt.

```
moveForward(100);
turnRight(90);
moveForward(100);
turnRight(90);
moveForward(100);
turnRight(90);
moveForward(100);
turnRight(90);
```

Hoeveel keer komt het aangeduide stuk terug in de code: _____

Hoe zou jij dit algoritme verbeteren?

_____

_____

Stap 5: Vierkant mét herhaling

Omdat hetzelfde stukje code in ons algoritme exact vier keer terugkomt, kunnen we beter een begrensde herhaming gebruiken. Dat is een herhaling waarvan wij, de programmeurs, exact weten hoeveel keer de acties uitgevoerd moeten worden. Hier is dat dus vier keer.

In blokken ziet ons algoritme er als volgt uit:

Het aangeduide deel ziet er vrij eenvoudig uit, maar wanneer we in de syntax duiken, zal je merken dat de begrensde herhaling meest ingewikkelde code is in deze syllabus. Let dus goed op!

Stap 6: Drie delen van een for-lus

Bekijk onderstaande code die hoort bij deze oefening:

```
for (var teller = 0; teller < 4; teller ++) {

    moveForward(100);
    turnRight(90);

}
```

De eerste regel bevat drie delen oftewel **drie parameters**. Die geven onze **begrensde herhaling** alle informatie die het nodig heeft om de acties te herhalen en op tijd te stoppen. Die **parameters** zijn telkens gescheiden van elkaar door een puntkomma.

Vul onderstaande tabel in:

Code	Functie	Uitleg
var teller = 0		
teller < 4		
teller++		

Bekijk volgende code en beantwoord de bijhorende vragen:

```
for (var teller = 0; teller < 5; teller ++) {

    moveForward(150);
    turnLeft(72);

}
```

De teller start bij: _____

De teller stop bij: _____

De code wordt X keer uitgevoerd: _____

De draaihoek bedraagt: _____

We bekomen volgende figuur: _____

Stap 7: While of for? Herhalingen leren kiezen.

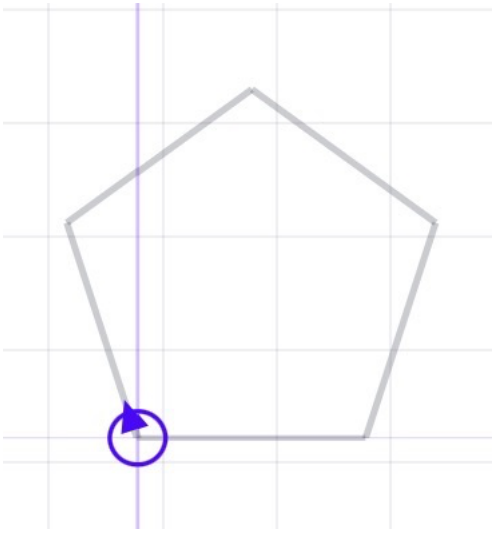
Je kent nu twee soorten herhalingen: de **voorwaardelijke herhaling** en de **begrensde herhaling**.

Soort	Syntax	Gebruik ik als …
Voorwaardelijke herhaling		
Begrensde herhaling		

Level 2: vijfhoek tekenen

Stap 1: Hoeken afleiden

In het vorige level tekenden we een vierkant. In dit level tekenen we een vijfhoek.



Bekijk de figuur en los de bijhorende vragen op:

Hoeveel zijden en hoeken telt onze vijfhoek?

_____

Wat kan je daaruit afleiden voor onze begrensde herhaling?

_____

_____

Duid op de figuur een binnenhoek α aan. Welk soort hoek is dit? scherp / recht / stomp

Gegeven: De binnenhoek α van de vijfhoek is 72 graden.

De binnenhoek en de draaihoek zijn elkaars _____.

De som van beide hoeken bedraagt dus _____ graden.

De draaihoek is dus _____ graden.

Stap 2: Algoritme ontwikkelen

Gebruik de code uit de vorige oefening als geheugensteuntje en noteer in het kader hieronder jouw blokkencode en daarna jouw JavaScript-code om een vijfhoek te tekenen.

```
for (var teller = 0; teller < 4; teller ++) {

    moveForward(100);
    turnLeft(90);

}
```

Blokkenschema	Codetaal

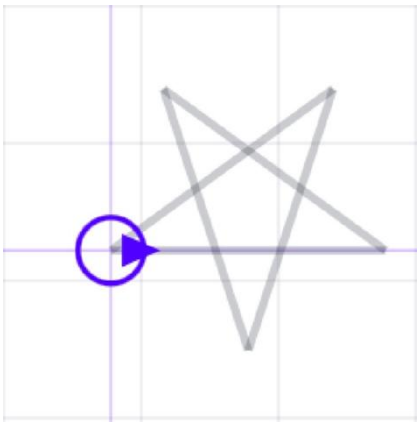
Duid in jouw JavaScript-code met kleur de parameters aan die verschillen met de code voor het vierkant. Welke stukjes heb je veranderd?

Stap 3: Daag jezelf uit!

Ga de uitdaging aan om in het webplatform level 1 en level 2 opnieuw te maken zonder te spieken in de syllabus. Slaag je er in om de ingewikkelde syntax van de for-lus met zijn drie parameters (start, eind, interval) uit het hoofd neer te schrijven?

Door jezelf telkens uit te dagen om leerstof uit het geheugen op te roepen, versterk je de verbindingen tussen leerstofonderdelen in jouw hersenen. Hoe sterker en talrijker de verbanden, hoe beter je iets kan onthouden. Leertip!

Level 3: Ster tekenen



Gewapend met de kennis en code om een vijfhoek te tekenen, kan je heel eenvoudig een ster tekenen. De verschillen tussen beide algoritmes zijn heel klein.

Beantwoord volgende vragen en ga meteen aan de slag met de JavaScript-code:

Onze ster telt _____ lijnen.

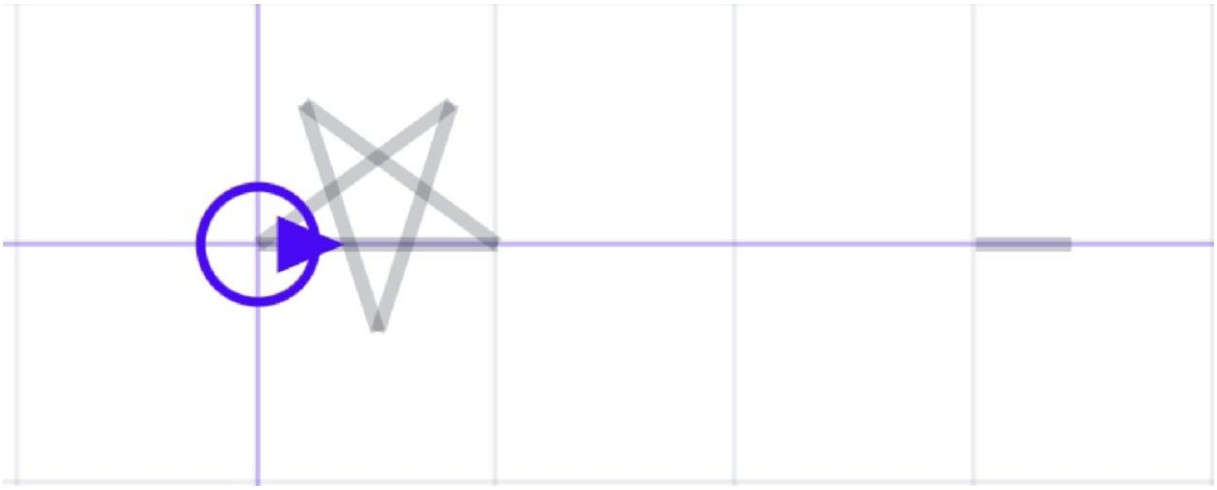
Om deze ster te tekenen draaien we naar:

_____

Gegeven de binnenhoek α die 36 graden is, dan is de draaihoek β _____ graden want

beide hoeken zijn elkaars _____

Level 4: Up & down



Bekijk de figuur die hoort bij dit nieuwe level. Wanneer we deze figuren met de hand willen tekenen, merken we dat we twee nieuwe acties nodig hebben om dit aan de computer duidelijk te maken.

Welke acties ontbreken er in ons huidig repetoire? _____

Je hebt reeds flink wat syntax gezien in onze oefeningen.
Maak een voorspelling: hoe zou de syntax voor die beide acties eruit zien?

_____

Zitten die twee acties in onze begrensde herhaling of niet? Motiveer jouw antwoord.

_____

_____

Level 5: Geneste herhalingen

Stap 1: Decompositie

Deze oefening ziet er ingewikkeld uit, maar eigenlijk bevat het enkel uitdagingen die je al kan oplossen. Alleen zal je hier met meerdere herhalingen moeten redeneren. Bij elke grote opdracht beginnen we met de allereerste stap uit het computationeel denken: de decompositie!

We delen de opdracht op in volgende stappen:

1) _____

_____

2) _____

_____

3) _____

_____

In welke vorm staan deze sterren? _____

Stap 2: Eén ster tekenen

Gegeven:

```
for (var teller = 0; teller < 5; teller ++) {
    moveForward(50);
    turnLeft(144);
}
```

Beantwoord volgende vragen:

De teller heet hier: _____

De lus herhaalt zich_____ **keer.**

Als draaihoek α is _____**graden is, dan is de binnenhoek β** _____**graden.**

α en β zijn elkaars: _____

Stap 3: Springen naar de volgende ster

Na één ster te hebben getekend moeten we springen naar de volgende plaats zonder een overbodige lijn te tekenen op ons scherm.

Vul onderstaande code aan als je weet dat de afstand tussen twee sterren telkens 150 pixels bedraagt.

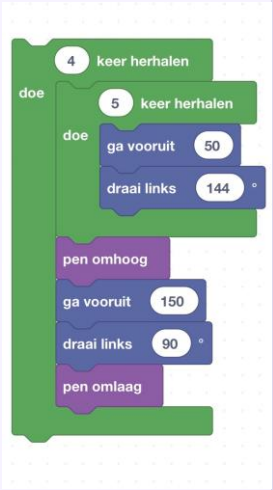
```
for (var teller = 0; teller < 5; teller ++) {
    moveForward(50);
    turnLeft(144);
}
```

Stap 4: Nesten

We willen in totaal vier sterren tekenen. Elke keer doen we dus volgende opdrachten:

- Teken één ster
- Verplaats de schildpad

Maar teken één ster bevat zelf ook al een herhaallus. Daarom landen we hier dus bij een oplossing die een herhaling in een herhaling bevat. We noemen dit een **geneste herhaling**. Bij dit level heb je dus twee **for-lussen** nodig.

	For-lus	Wat doet de lus?	Hoe vaak?
	**Buitenste lus**		
	Binnenste lus		

Stap 5: Twee tellers gebruiken

Elke **for-lus** heeft een eigen, unieke, teller nodig want ze houden beiden iets anders bij. Doe je dit niet, dan zal jouw for-lus te snel ten einde zijn.

Dus niet:

```
for (var teller = 0; teller < 5; teller ++) {
    for (var teller = 0; teller < 5; teller++) {
        moveForward(X);
        turnLeft(Y);
    }
}
```

Merk op: door de inspringing of indent kan je netjes zien welke lus de geneste herhaallus is.

Hierboven zie je dat alle variabelen dezelfde naam hebben. We gebruiken in lus 1 de variabele teller om het aantal herhalingen bij te houden … en doen dan hetzelfde in de tweede lus. Wanneer je werkt met meerdere herhalingen die elk iets anders bijhouden, geef je hen unieke namen.

Dus wel:

```
for (var teller1 = 0; teller1 < 5; teller1 ++) {
    for (var teller2 = 0; teller2 < 5; teller2++) {
        moveForward(X);
        turnLeft(Y);
    }
}
```

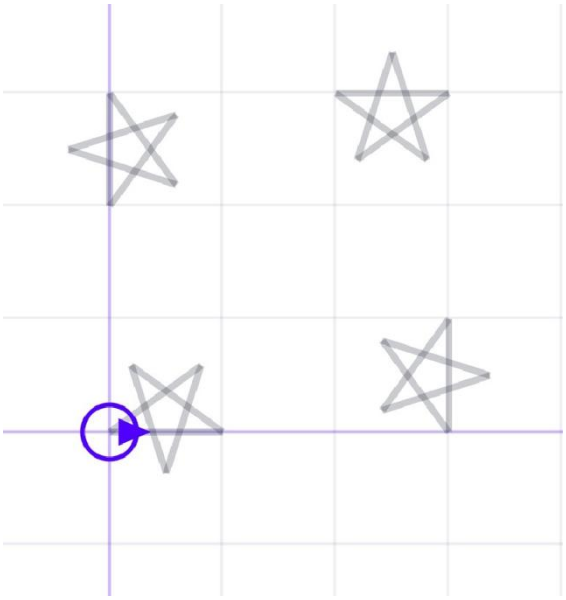
Vul aan:

For-lus	Wat doet de lus?	Hoe vaak?
teller1		
teller2		

Stap 6: Volledig algoritme ontwerpen

Werk hieronder het volledige algoritme uit voor de oefening. Daag jezelf uit en spiek niet naar vorige bladzijden bij het schrijven van jouw JavaScript-code. Let op voor:

- **De afstand tussen de sterren bedraagt 150 pixels.**
- **De lengte van elke "zijde" van een ster bedraagt 100 pixels.**
- **De binnenhoek bedraagt 36 graden, dus dan is de draaihoek …**
- **Gebruik twee herhaallussen met elk een unieke teller.**
- **Gebruik indents of inspringingen om jouw code leesbaar te houden.**



```
Noteer jouw code hier …
```

Level 6 en verder: Verdieping

Je kent nu de belangrijkste Turtle-bouwstenen. I de volgende oefeningen leer je niet telkens een volledig nieuw programmeerconcept. Je leert vooral bestaande bouwstenen combineren, herkennen waar een patroon zit en je code zelfstandig controleren. De levels zijn dus telkens moeilijker te tekenen, maar je hebt alle bouwstenen geleerd om ze aan te pakken. Op ééntje na:

Stap 1: naar voor, naar achter!

Onze turtle kan niet alleen naar voor bewegen, maar kan ook naar achter bewegen. De correcte syntax daarvoor kunnen we eigenlijk deduceren of afleiden van de gekende code:

```
moveForward(X);
```

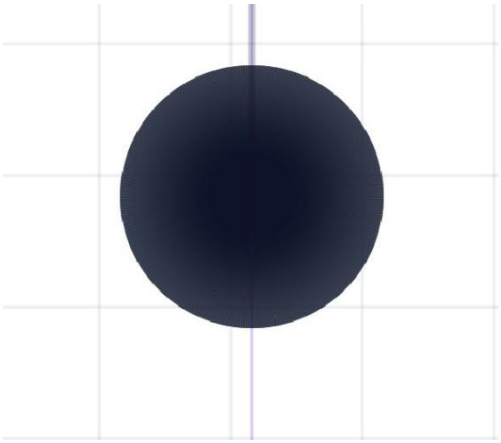
Dan is naar achter bewegen:

```

```

Stap 2: Full circle!

We weten uit eerdere levels dat complementaire hoeken samen 90 graden zijn en dat supplementaire hoeken 180 graden zijn. Tijd om voor de 360 graden te gaan, want in level 9 en 10 moet je namelijk een cirkel tekenen. Of eerder inkleuren!



Gegeven dat je start in het middelpunt van de cirkel. Hoe kan je deze inkleuren met blauw? Stel jezelf volgende vragen:

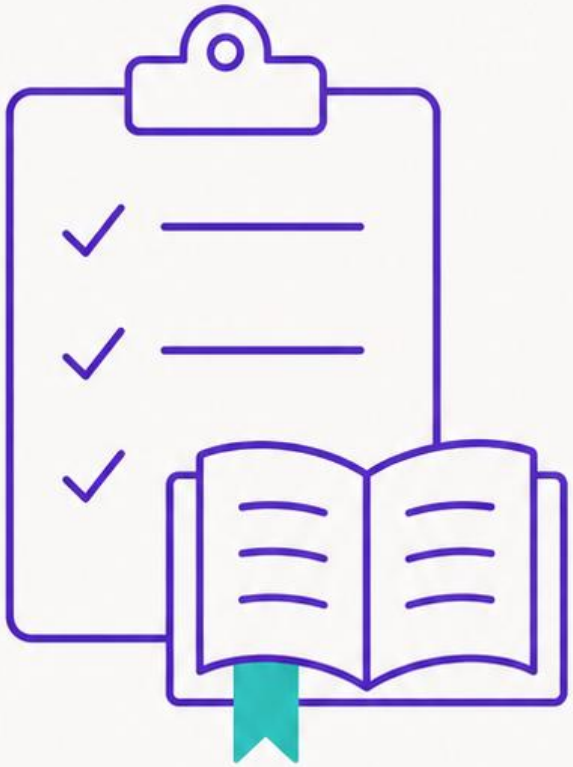
- **Welke twee acties voer ik uit?**
- **Hoeveel graden draai ik?**
- **Hoeveel keer herhaal ik?**

```
Noteer jouw code hier …
```

HOOFDSTUK 3

Leerdoelen
en begrippenlijsten

Een overzicht van wat je leert en de belangrijkste begrippen.



- Leerdoelen per onderwerp
- Begrippen en definities
- Formules en syntaxis
- Handig om te onthouden

Leerdoelen en begrippen

In dit onderdeel verzamelen we alle leerstof. Wat je moet kennen, hoe deze leerstof efficiënt kan studeren en hoe je jezelf daarbij kan controleren. Vink dus uit onderstaande niet zomaar alles aan. Vink pas een onderdeel aan wanneer je het ook echt kan uitleggen, toepassen of verbeteren.

Computationeel denken

- Ik kan een probleem opdelen in kleinere stappen. Dat is decompositie.
- Ik kan een patroon herkennen in een route, vorm of code.
- Ik kan alleen letten op wat belangrijk is voor de oplossing. Dat is abstractie.
- Ik kan een algoritme ontwerpen: een duidelijk stappenplan dat uitgevoerd kan worden.
- Ik kan een fout zoeken, verbeteren en uitleggen. Dat is debuggen.

Maze of doolhof

- Ik kan een eenvoudige route oplossen met blokken.
- Ik kan dezelfde route schrijven in JavaScript-code.
- Ik kan sequentie gebruiken: stappen in de juiste volgorde zetten.
- Ik kan een while-lus gebruiken wanneer iets moet herhalen zolang een voorwaarde waar is.
- Ik kan selectie gebruiken met if.
- Ik kan if en else gebruiken: als de voorwaarde waar is, gebeurt het eerste; anders gebeurt het tweede.
- Ik kan uitleggen wanneer een programma kan vastlopen in een oneindige lus.
- Ik kan voorspellen wat mijn Maze-code zal doen voor ik ze uitvoer.

Turtle of schildpad

- Ik kan een Turtle-tekening opdelen in kleinere onderdelen.
- Ik kan moveForward(...) gebruiken om de schildpad vooruit te bewegen.
- Ik kan turnRight(...) en turnLeft(...) gebruiken om de schildpad te draaien.
- Ik kan uitleggen wat een parameter is, bijvoorbeeld 100 in moveForward(100).
- Ik kan een draaihoek kiezen die past bij de figuur.
- Ik kan een for-lus gebruiken wanneer ik vooraf exact weet hoeveel keer iets moet gebeuren.
- Ik kan penUp() en penDown() gebruiken om te verplaatsen zonder of met tekenen.
- Ik kan een geneste herhaling herkennen: een herhaling binnen een herhaling.
- Ik kan twee tellers met verschillende namen gebruiken bij geneste for-lussen.

De vier programmeerconcepten

Concept	Wanneer gebruik je dit?	Voorbeeld
Sequentie	Wanneer stappen gewoon na elkaar moeten gebeuren.	`moveForward();` `turnRight();`
Voorwaardelijke herhaling	Wanneer iets moet herhalen zolang een voorwaarde waar is.	`while(notDone())`

Concept	Wanneer gebruik je dit?	Voorbeeld
Selectie	Wanneer het programma moet kiezen.	`if (isPathLeft())`
Begrensde herhaling	Wanneer je vooraf weet hoeveel keer iets moet gebeuren.	`for (var teller = 0; teller < 4; teller++)`

Begrippenlijst

Begrip	Uitleg
Algoritme	Een duidelijk stappenplan dat uitgevoerd kan worden.
Decompositie	Een probleem opdelen in kleinere delen.
Patroonherkenning	Herkennen wat terugkomt in een probleem, route, figuur of code.
Abstractie	Alleen letten op wat nodig is voor de oplossing.
Debuggen	Fouten zoeken, verbeteren en opnieuw testen.
Syntax	De schrijf of spellingsregels van code.
Sequentie	Instructies die stap voor stap na elkaar worden uitgevoerd.
Voorwaarde	Iets dat waar of niet waar kan zijn.
Selectie	Een keuze in je programma, meestal met if of else.
While-lus	Een herhaling die blijft lopen zolang een voorwaarde waar is.
For-lus	Een herhaling waarvan je vooraf weet hoeveel keer ze moet gebeuren.

Begrip	Uitleg
Parameter	Extra informatie tussen haakjes, bijvoorbeeld afstand of draaihoek.
Accolades { }	Tonen welke code bij een lus of keuze hoort.
Inspringing	Code naar rechts plaatsen zodat mensen de structuur beter kunnen lezen.
Oneindige lus	Een herhaling die blijft doorgaan omdat ze geen goed stopmoment bereikt.
Geneste herhaling	Een herhaling binnen een andere herhaling.
Teller	Een variabele die bijhoudt hoe vaak een for-lus al liep.

Hoe studeer ik deze leerstof?

Stap 1: Plan!

Plan meerdere korte studiemomenten doorheen de week. Eén lange sessie vlak voor een toets of examen werkt minder goed. Dit heet *"spaced practice"*, omdat je de oefenmomenten spreidt doorheen de tijd. *Blokken* werkt minder goed.

Stap 2: Studeer actief!

Lees de theorie niet alleen. Leg elk begrip kort uit in je eigen woorden. Hierbij kan je gebruikmaken van bijvoorbeeld steekkaartjes, hardop leren, jezelf (laten) opvragen.

Stap 3: Oefen schriftelijk

Oefen niet enkel met de blokken, maar schrijf de code ook herhaaldelijk op. Zo leg je verbindingen tussen jouw motorisch geheugen en de leerstof.

Stap 4: Test jezelf

Gebruik het leerplatform opnieuw. Maak enkele levels opnieuw zonder naar je oude oplossing te kijken.

Stel jezelf telkens volgende vragen:

1. Welk programmeerconcept heb ik hier nodig?
2. Wat voorspel ik dat mijn code zal doen?
3. Welke fout maakte ik en hoe heb ik die verbeterd?

Stap 5: Controleer jouw toetsen en oefeningen

Leerplandoelen

Deze syllabus sluit aan bij meerdere leerplandoelen uit ICT, STEM-wetenschappen en het Gemeenschappelijk Funderend Leerplan (KOV). De nadruk ligt in deze syllabus wel nadrukkelijk op computationeel denken, algoritmes ontwerpen, programmeren met blokken en JavaScript, testen, debuggen en reflecteren op het eigen leerproces. Onderstaande tabel is onderhevig aan veranderingen bij de leerplannen zelf. Controleer zelf telkens de leerplandoelen en nummers.

Leerplan	Leerplandoel	Koppeling binnen deze syllabus
ICT 1ste graad A/B-stroom	LPD 8: De leerlingen ontwerpen doelgericht een digitaal en niet-digitaal algoritme volgens de principes van computationeel denken en debuggen het.	Dit is het kernleerdoel van de syllabus. Leerlingen analyseren Maze- en Turtle-opdrachten, delen problemen op, herkennen patronen, ontwerpen algoritmes, testen hun oplossing en debuggen fouten.
	LPD 5: De leerlingen gebruiken doelgericht basisfunctionaliteiten van toepassingen om digitale inhouden te creëren.	Leerlingen creëren digitale oplossingen in het Blockly-platform. In Maze bouwen ze werkende algoritmes; in Turtle creëren ze visuele figuren en patronen met code. Deze oplossingen slaan ze op, exporteren deze uit de leeromgeving, bewaren deze op de eigen computer en/of dienen ze in via de elektronische leeromgeving bij de vakdocent.
	LPD 1: De leerlingen demonstreren basisvaardigheden bij het gebruik van digitale toepassingen.	Leerlingen gebruiken de interface van het leerplatform: oefenmodus kiezen, blokken slepen, code schrijven, uitvoeren, foutmeldingen interpreteren, feedback gebruiken, bestanden exporteren en beheren …
	LPD 2: De leerlingen navigeren functioneel op internet met een browser.	Leerlingen openen en gebruiken het online oefenplatform in de browser. Ze navigeren tussen Maze, Turtle, blokken, code en verschillende levels.

Leerplan	Leerplandoel	Koppeling binnen deze syllabus
ICT 1ste graad A/B-stroom	LPD 4: De leerlingen gebruiken doelgericht basisfunctionaliteiten van toepassingen om digitale inhouden te beheren.	Leerlingen kunnen hun werk exporteren naar PDF. Die export kan gebruikt worden als bewijsstuk, feedbackdocument of portfolio-item. Ze bewaren deze bestanden op een eigen device, in de cloud (OneDrive, Google Drive) of dienen deze in via de elektronische leeromgeving.
Basisoptie STEM-wetenschappen — 1ste graad A-stroom	LPD 20: De leerlingen onderzoeken een programma met verschillende variabelen en controlestructuren voor een betekenisvol algoritme in een STEM-context.	Leerlingen werken met controlestructuren zoals sequentie, selectie, if, else, while en for. Bij Turtle gebruiken ze tellers in for-lussen en leren ze waarom geneste lussen verschillende tellernamen nodig hebben.
	LPD 4: De leerlingen gebruiken doelgericht hulpmiddelen om te visualiseren, te beschrijven of te verklaren.	Het platform visualiseert algoritmes. In Maze zien leerlingen stap voor stap hoe hun algoritme wordt uitgevoerd. In Turtle zien ze hoe code wordt omgezet in lijnen, hoeken, vormen en patronen.
Gemeenschappelijk Funderend Leerplan	LPD 26: De leerlingen zetten metacognitieve leer- en regulatiestrategieën in om zich leerinhouden eigen te maken.	De syllabus gebruikt voorspellen, testen, vergelijken, debuggen en reflecteren. Leerlingen leren hun eigen fouten benoemen en gericht verbeteren.
	LPD 25: De leerlingen gebruiken school- en vaktaal.	Leerlingen gebruiken vaktaal zoals algoritme, decompositie, patroonherkenning, abstractie, debuggen, syntax, sequentie, selectie, voorwaarde, lus, parameter, accolade, teller en geneste herhaling.

Leerplan	Leerplandoel	Koppeling binnen deze syllabus
Gemeenschappelijk Funderend Leerplan	LPD 17: De leerlingen gebruiken doelgericht functionaliteiten van toepassingen om digitale inhouden te creëren.	Leerlingen maken digitale oplossingen met blokken en JavaScript. De Turtle-opdrachten maken dit zichtbaar via figuren en patronen; de Maze-opdrachten via werkende route-algoritmes.

Feedback, contact en licentie

Feedback op dit leermateriaal

Dit lesmateriaal is gemaakt om leerlingen stap voor stap te laten groeien van blokken naar JavaScript-cod en hen zo te ondersteunen in het leren computationeel denken.

Merk je tijdens het gebruik een fout, een onduidelijke opdracht, een technisch probleem of een oefening die beter kan? Een bug in mijn code? Noteer dan zo precies mogelijk wat er misloopt en laat het me weten. Debuggen is ook een deel van mijn opdracht als auteur van deze syllabus.

Vermeld bij feedback liefst:

- het hoofdstuk en/of pagina;
- het level;
- de oefening of opdracht;
- wat je verwachtte;
- wat er werkelijk gebeurde;
- welke browser of toestel je gebruikte;
- eventueel een schermafbeelding.

Goede feedback helpt mij om het leermateriaal duidelijker en bruikbaarder te maken voor alle leerlingen.

Contact

Dit materiaal werd ontwikkeld door:

Robbe Wulgaert
AI in de Klas
robbewulgaert.be

Voor vragen over dit lesmateriaal, het leerplatform of een mogelijke nascholing kun je contact opnemen via de contactpagina op robbewulgaert.be.

Wie lesmateriaal wil gebruiken of hierover wil uitwisselen met andere onderwijsprofessionals, kan ook aansluiten bij de community rond AI in de Klas.

Licentie en gebruik

Gebruik in de eigen klas of eigen onderwijscontext is toegestaan, op voorwaarde dat de auteur telkens duidelijk vermeld wordt.

Gebruik hiervoor deze naamsvermelding:

Robbe Wulgaert, AI in de Klas, robbewulgaert.be

Zonder voorafgaande toestemming is het niet toegestaan om dit materiaal of delen van dit materiaal te verspreiden, als eigen werk te publiceren, zonder bronvermelding te herwerken of gebruiken.