Computational Thinking

(onderdeel digitale geletterdheid - SLO)

De term is bedacht door Seymour Papert in 1980 en sluit goed aan op zijn gedachten over maken in het onderwijs. Binnen Computational Thinking worden er 11 verschillende 'concepten' onderscheiden. Deze concepten zijn deels gebaseerd op hoe computers werken, maar gaan vooral over het leren denken op een logische manier. Computational Thinking is dus ook niet, zoals vaak gedacht wordt, hetzelfde als programmeren.

Een definitie

Computational thinking (CT) is een proces dat een oplossing voor open problemen generaliseert. Open problemen vragen om volledige, betekenisvolle oplossingen, gebaseerd op meerdere variabelen. Computational thinking vereist het ontleden van het gehele beslissingsnemingsproces, de betrokken variabelen en alle mogelijke oplossingen, waardoor je verzekert bent dat de juiste beslissing genomen wordt op basis van de corresponderende parameters en beperkingen van het probleem. Computational thinking kan gebruikt worden om algoritmisch grootschalige problemen op te lossen en wordt vaak gebruikt om grote efficiëntieverbeteringen te realiseren.

Eenvoudig

Computationeel denken (of robot-denken) is een manier om problemen op te lossen. Dit kan een eenvoudig probleem zijn, zoals 'hoe kan ik een peer eten?' Of een ingewikkeld probleem, zoals 'hoe kan ik een wereldreis maken?'

Zo gezien zijn het dus vaardigheden die leerlingen helpen om een beter overzicht te krijgen van complexe problemen. Toch iets anders dan denken als een computer of leren programmeren. Maar lijkt Computational Thinking dan eigenlijk niet heel erg op het 'ouderwetse' redeneren? En wat als Computational Thinking een strategie is om problemen op te lossen, waarom staat het dan in het nieuwe 21st century skills model van Kennisnet en SLO?

Computationeel denken bevat vaak de volgende onderdelen

- Goed nadenken over alle informatie.
- Informatie in logische stukjes verdelen.
- Een schema of tekening van de informatie maken.
- Informatie versimpelen.
- Mogelijke oplossingen bedenken en uitproberen.
- Oplossingen automatiseren door algoritmisch te denken (stroomschema).
- De oplossing algemeen maken en toepassen op soortgelijke problemen.

Vier groepen

Ontleding (Decomposition)

De eerste groep is 'ontleding'. Door grote problemen op te delen in kleine stukjes wordt het steeds eenvoudiger om het op te lossen. Dit is een nuttige vaardigheid voor leerlingen, want door stap voor stap kleine problemen op te lossen kun je uiteindelijk een groot probleem opgelost hebben.

Patronen herkennen (Pattern recognition)

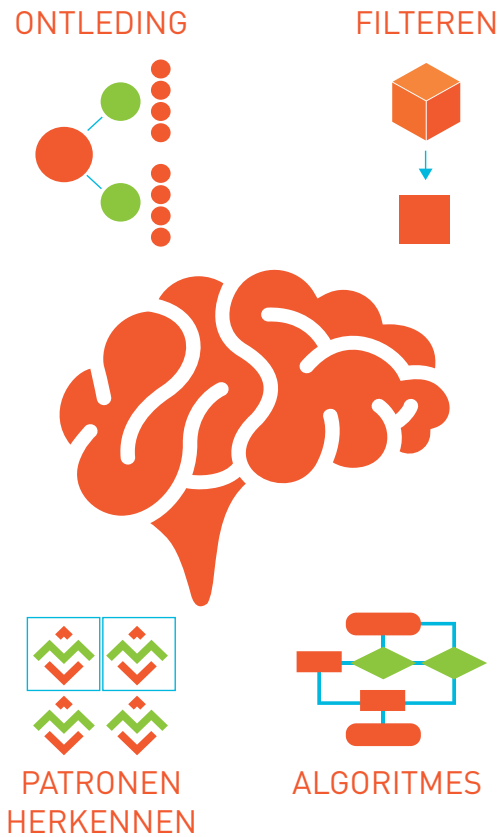
Wanneer een probleem ontleed is en opgedeeld is in verschillende stukjes, wordt het tijd om te kijken naar patronen. Door te zoeken naar verschillende onderdelen die op elkaar lijken kan een groot probleem eenvoudiger opgelost worden, omdat niet telkens hetzelfde gedaan moet worden.

Filteren (Abstraction)

Deze groep gaat over het filteren van de informatie. Wat is echt belangrijk? Wat raakt de kern van het probleem? Door te filteren krijgen leerlingen een beter idee van het probleem dat ze op proberen te lossen en kunnen ze effectiever aan de slag om het op te lossen.

Algoritmes (Algorithms)

De laatste groep gaat over de plan van aanpak: het maken van een algoritme. Door de te volgen stappen uit te werken kan een probleem opgelost worden. Dit kan natuurlijk alleen als het probleem helder is en opgedeeld is in verschillende onderdelen.



Verder inzoomen

Ontleding (Decomposition)

Als een probleem niet is ontleed dan is het moeilijker om op te lossen. Het is moeilijker om in één keer met veel verschillende fasen te werken dan een probleem in een aantal kleinere problemen op te delen en ze een voor een op te lossen. Door het probleem in kleinere delen op te splitsen, kan elk kleiner probleem gedetailleerder worden onderzocht.
Het begrijpen van een complex probleem is dus eenvoudiger door het op te delen. Het begrijpen van hoe een fiets met versnellingen werkt wordt eenvoudiger door de hele fiets op te delen in onderdelen of deelsystemen. Er is een versnelling, er zijn tandwielen, er is een overbrenging, er zijn remmen, de wielen hebben een bepaalde grootte, er is een frame die op een bepaalde manier is geconstrueerd enz.

Je bent dagelijks bezig met ontleden van problemen, vaak zonder dat je het in de gaten hebt. Bij het verzorgen van je tanden kies je een tandenborstel, de tandpasta, de duur van het poetsen, de volgorde, de druk die je zet bij het poetsen en je denkt misschien nog na of je wel of niet moet flossen.

Het oplossen van een misdaad (kijk maar eens naar CSI) gebeurt aan de hand van het ontleden van deelaspecten van wat er gebeurd is. Wanneer is het gebeurd? Waar is het gebeurd? Wie is het slachtoffer? Zijn er bewijsstukken? Welke relaties had het slachtoffer? Wat deed het slachtoffer voordat hij werd overvallen? Is dit al eens vaker gebeurd? Enzovoort.....

Het maken van een website of een app vraagt ook om 'ontleding'. Voor wie is de website? Wat is het doel van de website? Wat is de verhouding tekst-beeld? Welke beelden passen bij de doelstelling? Hoort er audio op de website? Moet de site getest worden? Hoe zorg ik dat de website wordt gevonden? Welke trefwoorden passen bij de website? Enzovoort....

Patronen herkennen (Patterns)

Patroonherkenning is een van de vier hoekstenen van de informatica. Het gaat om het vinden van de overeenkomsten of patronen tussen kleine, uitelkaar gerafelde problemen die ons kunnen helpen om complexere problemen efficiënter op te lossen.

Alle katten hebben gemeenschappelijke kenmerken. Ze hebben allemaal ogen een staart en een vacht. Ze eten graag vis en maken miauwende geluiden. Omdat we weten dat alle katten ogen, een staart en een vacht hebben, kunnen we een goede poging doen om een kat te tekenen, simpelweg door deze gemeenschappelijke kenmerken op te nemen.

In computational thinking staan deze kenmerken bekend als patronen. Als we eenmaal weten hoe we een kat moeten beschrijven, kunnen we andere katten beschrijven, gewoon door dit patroon te volgen. De enige dingen die anders zijn, zijn de details:

- een kat kan groene ogen, een lange staart en zwarte vacht.
- een andere kat kan gele ogen hebben, een korte staart en gestreepte pels.

Het vinden van patronen is heel belangrijk. Patronen maken een taak eenvoudiger. Problemen zijn gemakkelijker op te lossen als er gezamenlijke patronen zijn. We kunnen dezelfde probleemoplossende oplossing gebruiken als er eenzelfde patroon wordt herkend. Hoe meer patronen we kunnen vinden, hoe gemakkelijker en sneller het probleem opgelost kan worden. Stel dat we het patroon 'kat' niet goed hadden bekeken dan hadden we telkens moeten nadenken en checken hoe een kat eruitziet. Dat zou veel meer tijd kosten.

Het is van groot belang om op zoek te gaan naar patronen binnen een probleem en tussen problemen. Het bakken van een cake is een mooi voorbeeld van een patroon binnen een probleem. Als je het patroon kent van het bakken van een cake (ingrediënten, wijze van mengen, warmte oven, baktijd en bakvorm) dan is het heel eenvoudig om te gaan variëren.

Filteren (Abstraction)

Filteren is het proces van negeren van kenmerken van patronen die we niet nodig hebben om ons te concentreren op de dingen die we doen. Het is ook het filteren van specifieke details. Hieruit creëren we een idee van wat we proberen op te lossen.

Terug naar het voorbeeld van de katten. We hebben het patroon 'kat' ontdekt (ogen, vacht, staart, houd van vis en maakt miauwende geluiden). De ene kat heeft lange haren, de andere kat heeft groene ogen en de kat van de buurvrouw houdt van zalm. Dat zijn allemaal specifieke kenmerken. We hebben die kenmerken niet nodig om een kat te kunnen tekenen.

Abstractie stelt ons in staat om een algemeen beeld te vormen van wat het probleem is en hoe we het moeten oplossen. Het proces 'dwingt' ons om alle specifieke details en alle patronen te verwijderen die ons niet zullen helpen om het probleem op te lossen. Dit helpt ons om een idee van het probleem te vormen. Dit idee staat bekend als een 'model'.

Als we niet abstraheren, krijgen we misschien de verkeerde oplossing voor het probleem dat we proberen op te lossen. Als onze kat bijvoorbeeld niet abstraheert, denken we misschien dat alle katten lange staarten en een korte vacht hebben. Als we geabstraheerd zijn, weten we dat hoewel katten staarten en een vacht hebben, niet alle staarten lang zijn en niet alle vacht kortharig is. In dit geval heeft abstractie ons geholpen een duidelijker model van een kat te vormen.

Als we de informatie over het bakken van een cake filteren dan blijven er een paar wezenlijke dingen over. Je weet dat bij het maken van een cake ingrediënten nodig zijn, dat de ingrediënten in afgestemde hoeveelheden moeten worden afgemeten en dat een cake een bepaalde tijd in de oven moet (bij een bepaalde temperatuur).

Door informatie en gegevens te filteren creëer je een abstract model van bijvoorbeeld een kat of een cake. Dat model representeert alle katten en cakes. Kijk maar eens naar de plaatjes. Je weet direct waarover het gaat ook al is er heel veel informatie weggelaten. Zo'n tekening noem je een model.

Algoritmes

Een algoritme is een plan, een reeks stapsgewijze instructies om een probleem op te lossen. Als je veters kunt strikken, een kopje thee kunt zetten, je aankleden of een maaltijd kunt bereiden, dan weet je al hoe je een algoritme moet volgen.

In een algoritme wordt elke instructie geïdentificeerd en de volgorde waarin ze moeten worden uitgevoerd, is gepland. Algoritmes worden vaak gebruikt als startpunt voor het maken van een computerprogramma en ze worden soms geschreven als een stroomdiagram of een pseudocode.

Als we een computer willen vertellen iets te doen, moeten we een computerprogramma schrijven dat de computer stap voor stap vertelt wat we precies willen en hoe we het willen doen. Dit stap-voor-stap-programma moet worden gepland en hiervoor gebruiken we een algoritme.

Computers zijn alleen zo goed als de algoritmes die ze krijgen. Als je een computer een slecht algoritme geeft, krijg je een slecht resultaat - vandaar de zin: 'Garbage in, garbage out.'

Algoritmes worden voor veel verschillende dingen gebruikt, waaronder berekeningen, gegevensverwerking en automatisering.

Een pseudocode is een voorbeeld van een algoritme maar is geen programmeertaal. Een pseudocode wordt vaak gebruikt voordat er echt wordt geprogrammeerd. Hieronder een voorbeeld.

Output: Hoe heet je?
Input: De naam wordt ingevoerd.
Output: Welkom + naam
Output: Hoe oud ben je?
Input: De leeftijd wordt in jaren ingevoerd.
Output: Kleiner dan 18 - Deze site is echt iets voor jou.
Groter dan 18 - Ben je begeleider, leerkracht of ouder?