Programmeren in de bovenbouw van het basisonderwijs

Gerard Dummer

Inleiding

Dit artikel is bedoeld om je inzicht te geven in de manier waarop je met leerlingen van de bovenbouw van het basisonderwijs een programmeerles kunt geven in een blokgebaseerde programmeeromgeving. Onderwerpen die daarbij aan bod komen zijn: digitale geletterdheid, computational thinking, programmeerconcepten en didactische aanpak.

Gerard Dummer
Amersfoort, 8 mei 2020

Update: april 2021.
Update: 23 april 2021

Inhoudsopgave

INLEIDING **2**

INHOUDSOPGAVE **3**

1 DIGITALE GELETTERDHEID **4**

2 COMPUTATIONAL THINKING **5**

3 CT BUITEN HET PROGRAMMEREN **7**

4 CT EN PROGRAMMEREN VOOR KINDEREN **9**

5 WAAROM PROGRAMMEREN **10**

6 PROGRAMMEREN **11**

6.1 TEKSTGEBASEERD VS BLOKGEBASEERD 11
6.2 DOELEN PROGRAMMEREN 12
6.3 VOORBEELDEN VAN PROGRAMMEERCONCEPTEN IN SCRATCH 14

7 DIDACTISCHE AANPAK **15**

7.1 DE INTRODUCTIELES OF SPEELLES 15
7.2 DE PROGRAMMEERLES 20

8 EEN VOORBEELD **22**

8.1 INTRODUCTIE 22
8.2 UNPLUGGED ACTIVITEIT 23
8.3 GEBRUIKEN 25
8.4 AANPASSEN 26
8.5 CREËREN 26
8. 6 EVALUATIE 26

9. UNPLUGGED ACTIVITEITEN **27**

9.1 SEMANTIC WAVE 27
9.2 VOORBEELDEN UNPLUGGED ACTIVITEITEN 28

10 SOORTEN VRAGEN EN OPDRACHTEN **31**

10.1 VRAGEN OP BASIS VAN HET BLOCK-MODEL 31
10.2 OPDRACHTEN VAN BAGGY 33

11 VERDER DAN 1 LES **35**

11.1 KOPPELING MET VAKGEBIED EN VAKOVERSTIJGENDE DOELEN 35
11.2 AANLEREN PROGRAMMEERCONCEPTEN 36
11.3 LEREN WERKEN MET ONTWERPEISEN 36
11.4 VOORBEELD LESSENSERIE 36

12 DEBUGGEN **37**

13 LESIDEEËN EN ACHTERGRONDINFORMATIE **38**

14 LITERATUURLIJST **40**

BIJLAGEN **41**

BIJLAGE A – KENNIS OVER DE COMPUTER 42
BIJLAGE B – COMPUTER IN ANDERE APPARATEN 48

1 Digitale geletterdheid

Het is een open deur en een cliché om te zeggen dat we omringd worden door digitale technologie en dat het ons dagelijks leven beïnvloedt. Toch is het zo. Thuis, op school en in de maatschappij kom je digitale technologie tegen. Soms is die digitale technologie heel duidelijk aanwezig, bijvoorbeeld als je werkt op een laptop. Vaak is die digitale technologie verstopt en maken we er gebruik van zonder het door te hebben, bijvoorbeeld als je de wasmachine aanzet of televisiekijkt. Maar ook als je weet dat je van digitale technologie gebruik maakt, bijvoorbeeld omdat je op een website naar informatie zoekt of je sociale mediakanalen checkt, kan het zijn dat je je niet bewust bent wat die digitale technologie met je doet. Dat er bijvoorbeeld cookies je internetgedrag in de gaten houden en dat je advertenties te zien krijgt op basis van je likes en klikgedrag online.
Digitale technologie biedt ook nieuwe mogelijkheden om jezelf te ontwikkelen en te uiten. Digitale foto- en videocamera's (op een smartphone of anders) in combinatie met bewerkingssoftware maken het mogelijk om jezelf eenvoudig te presenteren of creatief te uiten. Online kun een groot publiek bereiken met je creatieve uitingen, sociale evenementen en maatschappelijke betrokkenheid.

Om leerlingen bewust en bekwaam te maken op het gebied van digitale technologie wordt er op school aandacht besteed aan digitale geletterdheid. Digitale geletterdheid (SLO & Kennisnet, 2016) bestaat uit vier componenten: ICT-basisvaardigheden, mediawijsheid, informatievaardigheden en computational thinking (zie Figuur 1).

Figuur 1: Model van Kennisnet en SLO waarin de relatie tussen 21ste eeuwse vaardigheden en digitale geletterdheid duidelijk wordt gemaakt.

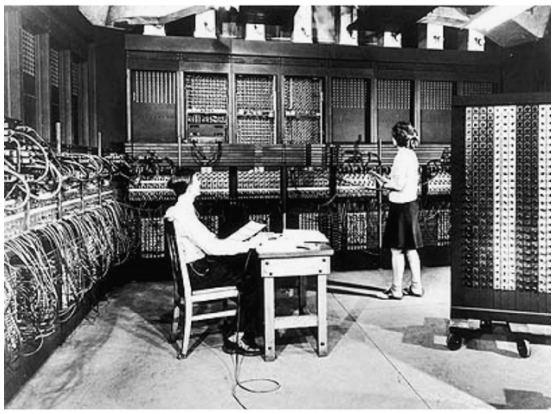
Bij ICT-basisvaardigheden gaat het om kunnen omgaan met de computerapparaten en randapparatuur zoals WORD, PowerPoint, browser printer, scanner en foto- en videocamera. Mediawijsheid gaat om het bewust, kritisch en actief kunnen omgaan met de gemedialiseerde samenleving (Raad voor Cultuur, 2005). Media biedt je mogelijkheden jezelf te ontwikkelen maar je wordt er ook door beïnvloed. Informatievaardigheden gaat over het leren omgaan met de grote hoeveelheid (digitale) informatie die tegenwoordig beschikbaar is. Hoe zoek je wat je vinden wilt en hoe weet je of wat je gevonden hebt ook betrouwbaar

is? Het nieuwste onderdeel van digitale geletterdheid is computational thinking. Bij computational thinking gaat het om het oplossen van problemen met behulp van de computer. Programmeren is daar nauw aan verbonden. In de volgende paragraaf gaan we hier dieper op in.

2 Computational thinking

Wing (2006) bracht de term computational thinking (CT) onder de aandacht omdat zij van mening is dat deze manier van problemen oplossen niet alleen nuttig was voor informatici bij het oplossen van programmeerproblemen maar voor iedereen nuttig zou zijn. In deze paragraaf beschrijven we CT met voorbeelden vanuit de informatica. In paragraaf 3 geven we voorbeelden van CT buiten het programmeren.
In 2006 omschrijft ze CT als het oplossen van problemen, het ontwerpen van systemen en het begrijpen van menselijk gedrag door voort te borduren op de fundamentele concepten van de informatica. Het gaat dus om het oplossen van problemen. Voor informatici zijn dat programmeerproblemen. Een informaticus gaat dus niet direct programmeren als hij of zij een probleem krijgt voorgelegd maar probeert het programmeerprobleem van tevoren te doordenken met een aantal denkvaardigheden.
Wat houden deze denkvaardigheden in? Shelby en Woollard (2013) en Angeli et al. (2016) komen tot de volgende vijf vaardigheden van CT: abstractie, generalisatie/ patronen, decompositie, algoritmisch denken en debuggen/ evalueren. *Abstractie* is de denkvaardigheid om te beslissen welke informatie over een onderwerp je moet behouden en welke je kunt negeren. Het helpt je omgaan met de complexiteit. Voor een informaticus is abstractie een handige denkvaardigheid, om ervoor te zorgen dat hij goed zicht houdt op de grote lijnen van de programmeercode. Stukken code die vaak terugkomen kan hij bijvoorbeeld in een apart deel van het programma stoppen (een routine) die hij weer op kan roepen als het nodig is.
Abstractie noemt Wing (2008) ook wel “our mental tool”. Als we geen onderscheid kunnen maken tussen hoofd- en bijzaken dan houden we waarschijnlijk geen overzicht en is nadenken dus niet zo gemakkelijk. Ze voegt daaraantoe dat “automation our metal tool” is. Met “automation” bedoelt ze de automatisering en daarmee wordt de rekenkracht van de computers bedoeld. Ze eindigt deze opsomming met de uitspraak dat informatica “the automation of our abstractions” is. Met computers kunnen allerlei problemen volgens Wing sneller worden opgelost. Een mooi voorbeeld waarin abstracties en automatisering samenkomen is het weerbericht. Een meteoroloog gebruikt allerlei formules die het gedrag van het weer proberen te vangen (de abstracties). Samen met de waarnemingen van heel veel weerstations wordt een voorspelling gedaan van het weer van morgen en de rest van de week. Al die gegevens berekenen zou voor een persoon onmogelijk zijn. Zelfs de eerste computer deed over het berekenen van het weersbericht van morgen vierentwintig uur. Nu berekent een supercomputer (zie Figuur 2) wat het weer wordt voor de komende week (de automatisering).

Figuur 2: Computers door de jaren heen die het weer proberen te voorspellen

De eerste computer die het weer probeerde te voorspellen (ENIAC). 	De supercomputer van het KNMI dat 58,2 biljoen (een miljoen x een miljoen) berekeningen per seconde kan maken. 

De denkvaardigheid *generalisatie* of *patronen* is het kunnen formuleren van een oplossing in algemene termen zodat het kan worden toegepast op een ander probleem. Het gaat hierbij om het herkennen van gezamenlijke patronen en het delen van gezamenlijke kenmerken. Als een informaticus een nieuw programmeerprobleem moet aanpakken is het voor hem handig om te weten of hij stukjes van het probleem al een keer ergens anders tegen is gekomen. Dat helpt bij het oplossen van het nieuwe probleem zodat je het wiel niet steeds opnieuw hoeft uit te vinden.
Decompositie is de denkvaardigheid om een complex probleem op te delen in kleinere delen die gemakkelijker te begrijpen en op te lossen zijn. Bij het schrijven een programma is het voor de informaticus handig om de verschillende deelaspecten te herkennen van het probleem en die op te lossen. Stel dat er een videobewerkingsprogramma gemaakt moet worden dan kan dat probleem opgedeeld worden in de verschillende onderdelen waaraan het programma moet voldoen (bijvoorbeeld het importeren van media, het bewerken van media en het exporteren van de media tot film). Aan de verschillende onderdelen van het probleem kan dan apart gewerkt worden.
Algoritmisch denken is de denkvaardigheid om een stappenplan op te stellen met activiteiten die je moet ondernemen om problemen op te lossen. Het gaat er daarbij om dat de activiteiten in de juiste volgende worden geplaatst en dat acties in de juiste volgorde worden uitgevoerd. Algoritmisch denken levert daarmee een algoritme op. Hoe slimmer het algoritme is opgesteld hoe sneller een probleem opgelost kan worden. Computer zijn snel maar met alleen brute force (een algoritme dat stap voor stap alle mogelijkheden afloopt) kom je er niet. Een algoritme dat snel kan selecteren en kan elimineren heeft de voorkeur. Zeker als de taken groter en complexer worden.
Debuggen is een denkvaardigheid die je in staat stelt om fouten te identificeren, te verwijderen en op te lossen. Een denkvaardigheid die daarmee samenhangt is evaluatie. Dat houdt in het kunnen analyseren of een programma goed werkt. Hieronder vallen testen en debuggen. Er zijn verschillende manieren waarop een informaticus een probleem kan identificeren. Er is zelfs software die hem hierbij ondersteunt.

3 CT buiten het programmeren

In paragraaf 2 zijn de verschillende denkstrategieën van CT toegelicht met voorbeelden uit de informatica zelf. Maar Wing (2006) gaf aan dat de verschillende denkvaardigheden van CT-denkvaardigheden zijn die voor iedereen interessant kunnen zijn. Vanuit verschillende kanten zijn daarbij ook voorbeelden gegeven over hoe dat er in het dagelijks leven uit zou kunnen zien. Een aantal voorbeelden op een rijtje.

Abstractie kom je ook in je dagelijks leven tegen. Er zijn heel veel voorbeelden waarin informatie wordt wegelaten om het overzichtelijk te maken. Bijvoorbeeld in de vorm van een lesrooster die je vertelt wat je die dag voor lessen krijgt (details over hoe je moet lopen of hoe de klas eruitziet, zijn weggelaten). Of een plattegrond van een stad, die afhankelijk van de schaal, zaken die niet relevant zijn, weglaat (zie Figuur 3). Of als we willen nadenken over een bepaald onderwerp helpt het ons om de details weg te laten en ons te focussen op de hoofdzaken.

Figuur 3: twee plattegronden met verschillende schaal.

Plattegrond van deel van Utrecht Science Park	Plattegrond van Nederland
	

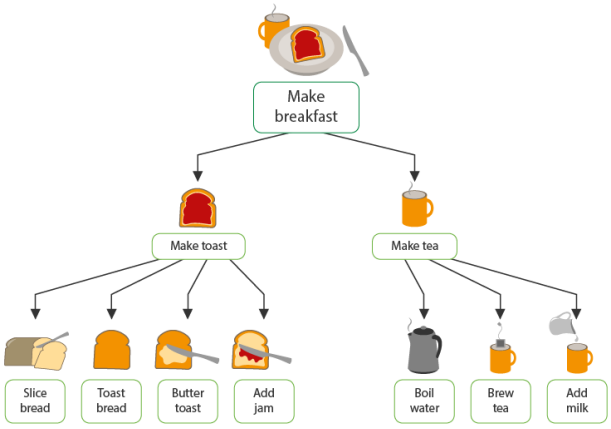
Generalisatie/ patronen zie je ook terug in het dagelijks leven. Bij het oplossen van een rekenprobleem bijvoorbeeld. Als je weet hoe je de oppervlakte van een rechthoek kunt oplossen dan kun je die formule ook voor andere rechthoek gebruiken. Of als je weet waarom een voorwerp blijft drijven kun je voorspellen welke andere voorwerpen dat ook zullen doen (zie Figuur 4).

Figuur 4. Wat zinkt en wat blijft drijven?

Decompositie zie je ook terug in het dagelijks leven. Bijvoorbeeld als je een plan maakt om iets te gaan doen. Denk bijvoorbeeld aan het maken van een taart of op vakantie gaan. Zo'n plan kun je opdelen in kleinere delen (voor het maken van een taart bijvoorbeeld: welke

ingrediënten heb ik nodig, hoe voeg ik ze samen en hoe bak ik het of hoe maak ik een ontbijt (zie Figuur 5) zodat elke taak afzonderlijk af te handelen is.

Figuur 5: voorbeeld van decompositie van het ontbijt.



Algoritmisch denken is ook een denkvaardigheid die je in het dagelijks leven terug ziet komen. Bijvoorbeeld als je een stappenplan volgt om een bepaalde taak op te lossen (een recept voor een gerecht (zie Figuur 6) of het in elkaar zetten van een IKEA-meubel) of als je zelf een stappenplan bedenkt voor een probleem.

Figuur 6: Recept voor pannenkoeken

Beslag

1. Doe het meel in een kom met hoge rand zodat je makkelijk kan mixen. Een maatbeker van 1(L) met giettuit is hiervoor geschikt.
2. Voeg de 2 eieren, de vanillesuiker, het zout en de helft van de melk toe.
3. Begin te mixen en voeg geleidelijk de rest van de melk toe.
4. Voeg als laatste een soeplepel maïskiemolie toe.
5. Mix nog even verder tot het beslag mooi homogeen wordt.

Het bakken

1. Gebruik liefst een pan met lage rand en een anti-kleeflaag.
2. Zet de koekenpan op een middelmatig vuurtje en doe een scheut maïskiemolie in de pan.
3. Verdeel de olie. Op het ogenblik dat de olie begint te "golven" is de pan heet genoeg om het beslag in de pan te doen.
4. Giet het beslag gelijkmatig in het midden van de pan totdat het beslag tot op 2 cm van de rand van de pan komt. Hef daarna de pan aan één zijde op zodat het beslag de rand van de pannenbodem bereikt. Doe dit naar alle richtingen zodat de hele bodem van de pan met beslag bedekt wordt.
5. Laat de pan daarna op het vuur staan tot het beslag droog wordt.
6. Draai dan de pannenkoek om.
7. Laat deze zijde nog ongeveer één minuutje bakken.
8. De pannenkoek is nu klaar.

4 CT en programmeren voor kinderen

Volgens Wing (2006) zou CT dus een algemene denkvaardigheid moeten zijn. Resnick (National Research Council, 2010) geeft aan dat CT weliswaar verder gaat dan programmeren maar alleen zoals taalvaardigheid meer is dan schrijven. Hij geeft aan dat programmeren een belangrijke expressiemanier is en dat programmeren, zoals schrijven, een beginpunt kan zijn voor nieuwe manier van denken. Aandacht besteden aan CT zonder te programmeren is volgens Resnick dus een gemiste kans.
Maar wat kunnen we verwachten als het gaat om CT en programmeren voor kinderen? Lister (2016) geeft aan dit lastig is en dat alleen expertprogrammeurs dit zouden kunnen. Hij (2016) beschrijft in zijn theorie over leren programmeren de verschillende, neo-Piagetiaanse fasen. Die begint bij de sensomotorische programmeerfase, waarin de beginnende programmeur moeite heeft om de code regel voor regel te kunnen lezen en zich nog niet kan voorstellen dat een computer een machine is die alleen maar kan uitvoeren zoals het is geprogrammeerd.
De tweede fase is de pre-operationele fase waarin de lerende programmeur in staat is om de code regel voor regel te lezen. Ze kunnen inductieve gissingen maken wat een stukje code doet. In deze fase kan de lerende programmeur nog niet loskomen van de code zelf. Debuggen zullen pre-operationele programmeurs intuïtief doen. Lister (2016) geeft aan dat in deze fase alleen code geschreven moet worden als ze nauwlettend gevolgd worden. Te veel code schrijven, waarin fouten kunnen sluipen, demotiveert leerlingen om met programmeren aan de slag te gaan. Het maken van schema's om hun programmeeractiviteiten te ondersteunen is voor pre-operationele programmeurs volgens Lister (2016) nog erg lastig omdat ze vooral over code na kunnen denken als ze die regel voor regel bestuderen.
In de concreet operationele fase zijn beginnende programmeurs in staat om op een deductieve manier over hun code na te denken. In deze fase kunnen beginnende programmeurs doelbewust te werk gaan en schema's effectief gebruiken om over hun code na te denken. Tot slot de formeel operationele fase. Dit is de expertfase waarin op een logische en systematische manier wordt geredeneerd. Een essentiële vaardigheid als het gaat om debuggen. Lister (2016) geeft aan dat deze fase niet is weggelegd voor kinderen van het basisonderwijs.
De CT-denkvaardigheden doen een beroep op metacognitieve vaardigheden. Een vaardigheid die volgens Lister (2016) wat betreft programmeren vooral is weggelegd voor expertprogrammeurs. Het gebruiken van schema's (of modellen zoals benoemd bij abstractie) en debuggen zijn vaardigheden die pas in de concreet operationele fase goed tot hun recht zouden komen.

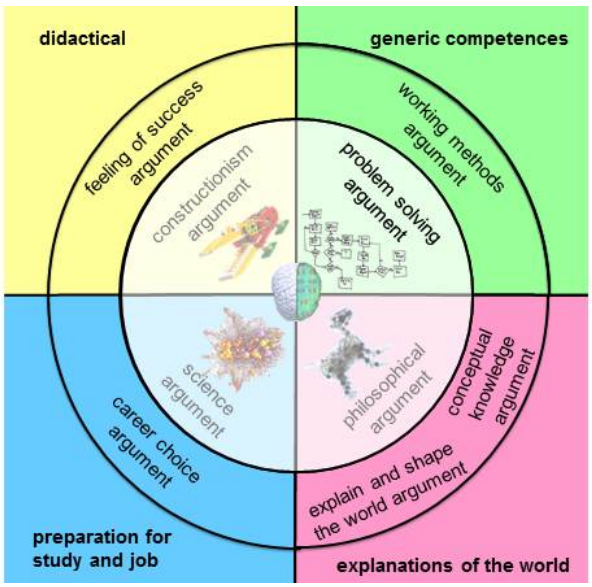
Betekent dat we dan geen aandacht zouden moeten besteden aan CT in het kader van programmeren met kinderen op de basisschool? Verschillende aspecten van CT kunnen wel aan bod komen maar we kunnen van leerlingen die nog moeten leren programmeren niet verwachten dat ze zelfstandig de verschillende CT denkvaardigheden inzetten. Ook moeten we goed voor ogen houden dat als we CT inzetten leerlingen niet opeens beter gaan programmeren. We kunnen wel CT inzetten omdat we het een waardevolle denkvaardigheid op zich vinden. Manieren waarop deze aan bod kunnen komen is door gezamenlijk op een geleide manier de volgende vragen te stellen en opdrachten te geven en deze ook gezamenlijk te beantwoorden en te maken:

Abstractie: Wat zijn de belangrijkste regels/ kenmerken.
Generalisatie/ patronen: Waar hebben we dat probleem eerder gezien? Wat was toen de oplossing?
Decompositie: Uit welke onderdelen bestaat dit?
Algoritmisch denken: Wat is een handige/ snelle manier om dit op te lossen?
Debuggen: hoe kunnen we de fouten opsporen?

5 Waarom programmeren

Er zijn verschillende redenen waarom je met leerlingen in de bovenbouw zou gaan programmeren. Het bedrijfsleven propageert programmeren vanuit een economisch perspectief: er zijn te weinig programmeurs om al het werk te kunnen doen. Ze voeren een stevige lobby om programmeren een plek te geven in het Nederlandse onderwijs. En hoewel het belangrijk is om goed opgeleide mensen te hebben die vacatures kunnen invullen in de Nederlandse economie, is dat alleen, geen goed uitgangspunt voor het integreren van programmeren in het basisonderwijs. Een mooi overzicht van uitgangspunten geeft Honegger (2015) in Figuur 7. Hij benoemt vier argumenten: science argument, philosophical argument, problem solving argument en constructionism argument.

Figuur 7: Uitgangspunten van Honegger voor programmeren



Het *science argument* is het argument dat door het bedrijfsleven wordt aangedragen. Het bereidt je voor op een vervolgstudie en beroep. Het *philosophical argument* zegt dat programmeren een manier is om de wereld om ons heen te begrijpen en mede vorm te geven. Je leert nieuwe belangrijke concepten kennen die je inzicht geven in de wereld om je heen. Met *problem solving argument* geeft Honneger aan dat programmeren nog een context is waarin je kunt oefenen met het oplossen van problemen. Tot slot het *constructionism argument*. Honneger geeft aan er leerlingen zijn die aanleg hebben voor programmeren en daarin succeservaringen kunnen opdoen.
Het uitgangspunt in dit hoofdstuk om met leerlingen te programmeren sluit aan bij de drie laatst genoemde argumenten. Daarbij zijn de vakken en het onderwijs zoals dat op de

basisschool gegeven wordt uitgangspunt en biedt programmeren een verdieping/ verbreding op die vakken.

6 Programmeren

We gaan wat dieper in op het onderwerp programmeren. We hebben het over tekstgebaseerd programmeren vs blokgebaseerd programmeren, de doelen van het programmeren en voorbeelden van programmeerconcepten in Scratch.

6.1 Tekstgebaseerd vs Blokgebaseerd

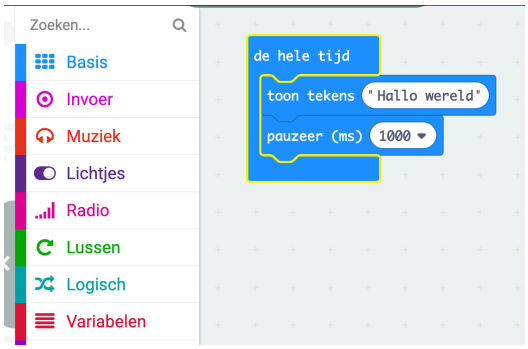
Uitgangspunt bij het programmeren in de bovenbouw van het basisonderwijs zijn de blokgebaseerde programmeeromgevingen zoals Scratch (https://scratch.mit.edu/) en MakeCode (https://www.microsoft.com/en-us/makecode). Voordeel van deze blokgebaseerde programmeertalen is dat syntaxis een minder grote drempel is ten opzichte van de tekstgebaseerde programmeertalen (zie Figuur 8). Een puntkomma vergeten of een haakje niet afsluiten of niet goed inspringen is bij de blokgebasseerde programmeertalen een minder grote drempel (zie Figuur 9) (Resnick et al., 2009).

Figuur 8: Tekstgebaseerde taal (Python) waarbij structuur (de syntaxis) gekleurd is.

```
def add5(x):
   return x+5

def dotwrite(ast):
   nodename = getNodename()
   label=symbol.sym_name.get(int(ast[0]),ast[0])
   print '    %s [label="%s' % (nodename, label),
   if isinstance(ast[1], str):
      if ast[1].strip():
         print '= %s"];' % ast[1]
      else:
         print '"]'
    else:
      print '"];'
      children = []
      for  in n, childenumerate(ast[1:]):
         children.append(dotwrite(child))
      print ,'    %s -> {' % nodename
      for  in :namechildren
         print '%s' % name,
```

Figuur 9: Screenshots met voorbeelden van blokgebaseerde programmeertalen

Voorbeeld van Scratch	Voorbeeld van MakeCode
	

Dat wil trouwens niet zeggen dat er met blokgebasseerde programmeetalen geen complexe programma's gemaakt kunnen worden. Resnick et al. (2009) citeert Papert die stelt dat programmeertalen een low floor moeten hebben (makkelijk om mee te beginnen) maar ook een high ceiling (doorgroeimogelijkheden voor iedereen die complexe projecten aan wil paken. Resnick et al. (2009) voegt daar het principe van wide walls aan toe. Het moet verschillende type projecten kunnen ondersteunen zodat iedereen met een andere interesse aan bod kan komen. Scratch biedt bijvoorbeeld op de website voorbeelden aan rondom spelletjes maar ook verhalen, animaties en muziek.

6.2 Doelen programmeren

Programmeren is een onderdeel van het vakgebied informatica. Informatica wordt als vak niet aangeboden in het Nederlandse basisonderwijs. In het buitenland, zoals Engeland en Amerika, zijn wel curricula ontwikkeld. In Engeland maakt het onderdeel uit van het vakgebied *computing*. De doelen van het Engelse Computing (CAS, 2013) zijn overzichtelijk en bieden daarom een mooi uitgangspunt om na te denken over de invulling van programmeren. Het vak computing is onderverdeeld in Computer Science (te vertalen met informatica), Information literacy (informatievaardigheden) en digital literacy (mediawijsheid). In tabel 1 staan voor de onderbouw (Key Stage 1) en bovebouw (Key Stage 2) de doelen voor de verschillende onderdelen van Computing.

Tabel 1: Doelen van het vak Computing in Engeland

	key stage 1 ages 5 - 7	**key stage 2 ages 7 - 11**
computer science	understand what algorithms are; how they are implemented as programs on digital devices; and that programs execute by following precise and unambiguous instructions create and debug simple programs use logical reasoning to predict the behaviour of simple programs	design, write and debug programs that accomplish specific goals, including controlling or simulating physical systems; solve problems by decomposing them into smaller parts use sequence, selection, and repetition in programs; work with variables and various forms of input and output use logical reasoning to explain how some simple algorithms work and to detect and correct error in algorithms and programs understand computer networks including the internet; how they can provide multiple services, such as the World Wide Web appreciate how [search] results are selected and ranked

information literacy	use technology purposefully to create, organise, store, manipulate and retrieve digital content	use search technologies effectively select, use and combine a variety of software (including internet services) on a range of digital devices to design and create a range of programs, systems and content that accomplish given goals, including collecting, analysing, evaluating and presenting data and information
digital literacy	recognise common uses of information technology beyond school use technology safely and respectfully, keeping personal information private; identify where to go for help and support when they have concerns about content or contact on the internet or other online technologies	understand the opportunities [networks] offer for communication and collaboration be discerning in evaluating digital content use technology safely, respectfully and responsibly; recognise acceptable/unacceptable behaviour; identify a range of ways to report concerns about content and contact

Omdat we ons in dit hoofdstuk richten op programmeren met kinderen in de bovenbouw van het basisonderwijs halen we deze doelen in tabel 2 nog een keer naar voren.

Tabel 2: Doelen Computer Science Key Stage 2

• Design, write and debug programs that accomplish specific goals, including controlling or simulating physical systems; solve problems by decomposing them into smaller parts • Use sequence, selection, and repetition in programs; work with variables and various forms of input and output • Use logical reasoning to explain how some simple algorithms work and to detect and correct error in algorithms and programs • Understand computer networks including the internet; how they can provide multiple services, such as the World Wide Web • Appreciate how [search] results are selected and ranked

Leerlingen in de bovenbouw moeten dus een computerprogramma kunnen maken met specifieke doelen, waaronder het aansturen fysieke systemen (zoals een motor of lichtje). Ze moeten leren om een probleem te op te delen in kleinere stukjes om het grotere probleem op te lossen (een CT denkvaardigheid trouwens: decompositie).
Dan zijn er nog een aantal programmeerconcepten die ze moeten kennen. Dat zijn sequentie (stapsgewijs iets uitvoeren), selectie (programmeerconcept ALS-DAN is dat), herhaling (of

lussen of herhaling). En ze moeten leren wat variabelen zijn (bijvoorbeeld in een spel waarin je een aantal levens hebt totdat je geraakt wordt door een tegenstander). Ook hebben ze geoefend met verschillende soorten input (bijvoorbeeld een spelcontroller maken met de Makey Makey) en verschillende soorten Output (bijvoorbeeld een motor of sensor van Lego WeDo).
Het gaat hier ook om logisch redeneren om simpele algoritmes uit te kunnen leggen en fouten hierin op te sporen.
Een beetje vreemde eend in de bijt is de aandacht voor computernetwerken en het internet, en de waardering voor de manier waarop zoekresultaten van een zoekmachine tot stand komen.

6.3 Voorbeelden van programmeerconcepten in Scratch

In paragraaf 6.2 werden verschillende programmeerconcepten genoemd. Er werd daarbij een hele korte omschrijving gegeven van wat het programmeerconcept precies inhield. In deze paragraaf zie je hoe zo'n programmeerconcept in Scratch eruit kan zien en wat het doet. In Scratch is elk programma gekoppeld aan een sprite. Standaard is dat het oranje katje (genaamd Scratch). De voorbeelden in tabel 3 zijn allemaal gekoppeld aan het katje Scratch.

Tabel 3: Voorbeelden van programmeerconcepten

Programmeerconcept	**In Scratch**	**Wat doet het?**
Sequentie		Het programma voert van boven naar beneden de verschillende programmeerblokken uit als er op de groene vlag wordt geklikt. Eerst neemt de kat 10 stappen. Dan wordt er één seconde gewacht. Tot slot zegt de kat 2 seconden "Hallo Wereld"
Herhaling		Precies zoals bovenstaande code. Maar er staat nu een herhaalblok om de vorige code. Dit zorgt ervoor dat de kat tien keer de bovenstaande code herhaalt. Als je het herhaalblok niet had gebruikt had je de code van sequentie 10 keer onder elkaar moeten zetten. Het herhaalblok kort de code dus flink in.

Selectie		Een selectie is een ALS DAN blok of een ALS DAN ANDERS blok. ALS je aan een voorwaarde voldoet DAN volgt er een actie. In dit voorbeeld zijn twee ALS DAN blokken. ALS pijltje rechts wordt ingedrukt DAN neemt Scratch 10 stappen. ALS pijltje links wordt ingedrukt DAN zegt Scratch “Hallo wereld”.
Variabele		Met een Variabele kan het computerprogramma iets onthouden. Een variabele heeft een naam en een waarde. In het voorbeeld is de naam van de Variabele Punten. De waarde is nul als je op de groene vlag klikt. Elk keer als je de voetbal raakt verandert de waarde van de variabele met één punt.
Lijst (of rij of reeks)		Een programmeerconcept dat niet op de Engelse lijst staat maar wel heel handig is, is het maken van een Lijst (array in het Engels). In een lijst kun je gegevens stoppen en je kunt ze ook uithalen. Elk gegeven krijgt een specifieke plek (plek 0, 1, 2, etc. Je kunt de gegevens weer oproepen uit de lijst door naar deze plek te verwijzen.

7 Didactische aanpak

Onze didactische aanpak bestaat uit twee delen. Het eerste deel is een introductieles of speelles en het tweede deel is de programmeerles.

7.1 De introductieles of speelles

Als leerlingen nog niet eerder hebben geprogrammeerd in de bovenbouw dan beginnen de leerlingen met een introductieles of speelles. Deze les bestaat uit een aantal onderdelen:

- de introductie-activiteit
- kennis laten maken met de programmeeromgeving
- experimenteren en
- evalueren.

Ad 1. Introductie-activiteiten

Als leerlingen nog geen goed beeld hebben van wat een computer is of nog nooit geprogrammeerd hebben, kun je leerlingen hier op verschillende manieren kennis mee laten maken. Je kunt hen kort informatie geven over wat een computer en programmeren is door middel van een presentatie of hen een opdracht geven waarmee ze ervaren wat een computer doet en wat programmeren is.

Over de computer

Leerlingen weten waarschijnlijk wel dat de apparaten die ze op school gebruiken (zoals de laptop, tablet, chrombook, dekstop) voorbeelden zijn van computers. Leerlingen weten nog niet altijd dat een computer ook in andere apparaten kan zitten zoals het kopieerapparaat op school of de wasmachine thuis of de verkeerslichten onderweg. Het antwoord op de vraag wat is een computer is dan ook niet een laptop of tablet. Dat zijn vormen waarin ze de computer kennen. Je kunt met leerlingen op verschillende manieren praten over wat een computer is. Bijvoorbeeld dat een computer bestaat uit hardware en software. Dat een computer input (informatie) nodig heeft, die moet verwerken zodat de computer weer output kunt geven. Je kunt ook met leerlingen praten over het feit dat computers in allerlei apparaten aanwezig is die ze in het dagelijks leven gebruiken. In bijlage A van dit artikel vind je uitleg over de belangrijkste hardware die in een computer zit en informatie over software. Zo kom je te weten waar de nullen en enen nu precies voor worden gebruikt en wat een hogere programmeertaal is. In bijlage B vind je informatie over de rol die de computer speelt in apparaten die je in het dagelijks leven kunt tegenkomen.

Over programmeren

Als leerlingen een beeld hebben van wat een computer is, is het voor hen gemakkelijker voor te stellen wat programmeren is. Kern van programmeren is het oplossen van problemen door het schrijven van een computerprogramma. Het computerprogramma bestaat uit instructies (algoritmes) die de computer kan uitvoeren. Zo'n instructie moet precies zijn.

Leerlingen die nog nooit hebben geprogrammeerd hebben verschillende vooronderstellingen over de manier waarop de computer werkt. Bijvoorbeeld dat de computer zelf wel zal begrijpen wat er bedoeld wordt als er een niet zo'n heldere instructie gegeven wordt. Mensen hebben immers soms aan een half woord genoeg. Computers echter niet. Die moet je precies vertellen wat ze moeten doen.

Type activiteiten ter introductie

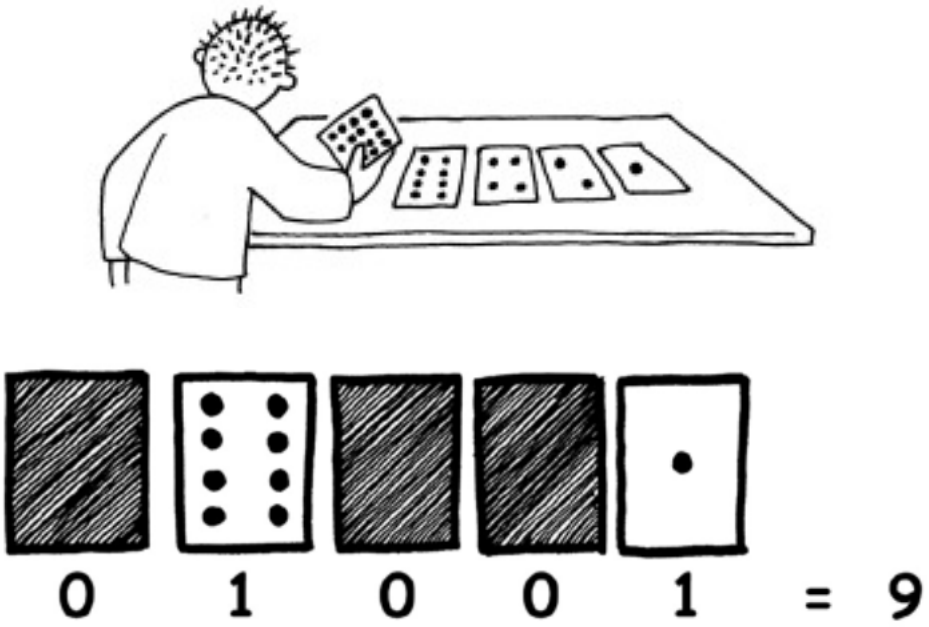
Je kunt een presentatie houden over de computer en een presentatie houden over programmeren. Hierin leg je uit wat een computer is en wat programmeren inhoudt. Je kunt ook opdrachten geven waarbij leerlingen ervaren wat de computer is of wat programmeren inhoudt. Deze opdrachten worden vaak zonder computer uitgevoerd. We noemen dit de outreach unplugged activiteiten. Over unplugged activiteiten om programmeerconcepten te introduceren lees je meer in paragraaf 9. De onderzoekers Bell, Freeman en Grimley (2009) ontwikkelden dit soort outreach activiteiten oorspronkelijk om buiten lessituaties om mensen in aanraking te brengen met computers en programmeren. Op de website https://csunplugged.org/en/ en in het Nederlands op http://www.csunplugged.nl/ vind je verschillende soorten activiteiten. Ook op de website van Codekinderen

(https://maken.wikiwijs.nl/100525/CodeKinderen) en Barefoot computing (https://www.barefootcomputing.org/) vind je van deze unplugged activiteiten.

Een paar concrete voorbeelden

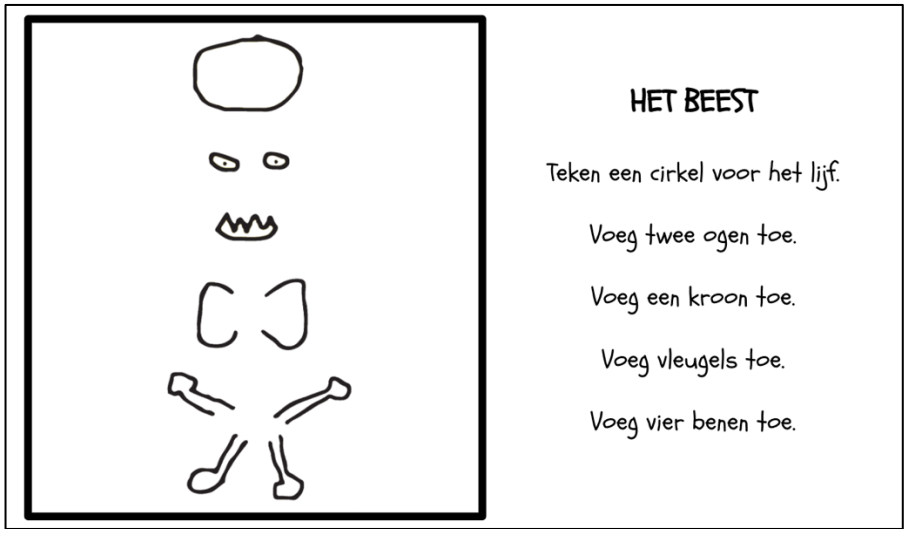
Binair tellen
Een voorbeeld om kinderen meer te leren over de manier waarop de computer werkt is het binair leren tellen. Op http://www.csunplugged.nl/introductie/deel-1/01-binaire-getallen/ staat een uitgebreide uitleg over hoe je zo'n activiteit kunt opzetten (zie Figuur 10).

Figuur 10: binair tellen. De plek van een nul of een geeft de waarde aan.



Het beest
Leerlingen bewust maken van het feit dat een computer hele precieze instructies nodig heeft om de opdracht uit te voeren is het tekenen van het beest (zie Figuur 11). Deze opdracht vind je uitgebreid beschreven op Barefoot computing (https://www.barefootcomputing.org/). Je geeft een voor een de instructies. De instructies worden waarschijnlijk op allerlei manieren uitgevoerd en leidt tot allerlei verschillende beesten. In de nabespreking van deze activiteit kun je aangeven dat de instructies nog niet precies genoeg waren. Je maakt daarna de vergelijking met een computer die niet zonder precieze instructies weet wat jij bedoelt.

Figuur 11. De vragen die je kunt stellen bij de unplugged opdracht: Het beest



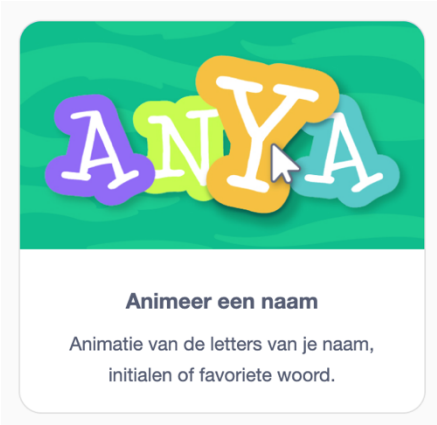
De hagelslagrobot
Leerlingen programmeren hun juf of meester met instructies die een boterham met hagelslag moet smeren (zie figuur 12). Deze activiteit staat uitgebreid beschreven op https://maken.wikiwijs.nl/100525/CodeKinderen#!page-3954584. Het idee van deze activiteit is dat over de volgorde van de opdracht goed moet worden nagedacht omdat de robot (computer) het anders niet goed kan uitvoeren.

Figuur 12: Still uit de video van unplugged activiteit van de hagelslagrobot

Ad 2. kennis laten maken met de programmeeromgeving
Nadat leerlingen een introductie hebben gekregen over de computer en/of programmeren kun je met de leerlingen daadwerkelijk gaan programmeren. Dit doe je door een project dat je hebt voorbereid te introduceren in de programmeeromgeving, de werking ervan te

demonsteren, de maakomgeving kort uit te leggen en de manier waarop je de code gemaakt te modellen.

Geschikte projecten om te gebruiken in de speelles
In deze speelles staat experimenteren centraal. Dat gaan leerlingen in de volgende fase van de les dan ook doen. Om ervoor te zorgen dat ze zich vooral kunnen concentreren op het experimenteren moet je een project aanbieden waarin leerlingen alleen maar het programmeerconcept sequentie hoeven te gebruiken. In deze fase is het goed om leerlingen te laten ervaren wat het effect is als verschillende programmeerblokken onder elkaar worden geplakt. Een voorbeeld van een eenvoudige programmeeractiviteit die gebruikt maakt van sequentie is het animeren van letters. Op de website van Scratch (https://scratch.mit.edu/ideas) vind je daar een voorbeeld van (zie Figuur 13).



Figuur 13

Demonsteren
Ter voorbereiding van de les heb je het type project waarmee je wilt dat leerlingen gaan experimenten klaar gezet. Bij het animeren van letters heb je bijvoorbeeld al de letters van je eigen naam in Scratch gezet en de eerste letter daarvan geanimeerd. Op de projectpagina demonstreer de werking van het project.

Maakomgeving uitleggen
Je laat nu de maakomgeving van Scratch zien (zie Figuur 14). Je legt daarbij uit wat de leerlingen zien. Deze instructie houd je zo kort mogelijk. Je geeft aan dat aan de linkerkant verschillende programmeerblokken staand die ze kunnen gebruiken, midden in beeld staat de gebruikte code, rechtsboven zien ze het project en rechtsonder staan de sprites. Belangrijk vooral is om te benadrukken dat elke sprite zijn eigen code krijgt.

Figuur 13. Screenshot van de maakomgeving van Scratch met animatie van letters

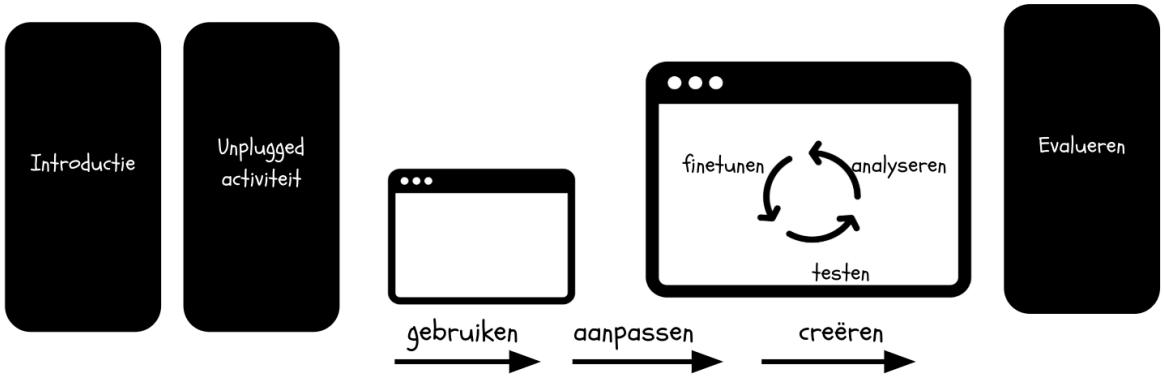
Modellen
Laat nu zien hoe je de code met de programmeerblokken in elkaar hebt gesleept. Hierbij gaat het er vooral om dat leerlingen zien hoe ze blokken naar het programmeerveld kunnen slepen, de blokken van volgorde kunnen veranderen en blokken kunnen weggooien.

Ad 3. experimenteren
De leerlingen gaan nu zelf experimenteren met het de programmeeromgeving. Het doel is om een soortgelijk project te maken. Leerlingen mogen in deze les het proberen om het zo uitbundig mogelijk te maken. Als leerkracht kun je rondlopen, leerlingen aanmoedigen en enthousiasmeren. Let erop dat er vragen kunnen komen van leerlingen waarvoor ze meer programmeerervaring nodig hebben. Het is niet de bedoeling om in deze les allerlei nieuwe programmeerconcepten uit te leggen in principe. Je kunt leerlingen wel wijzen natuurlijk op waarschijnlijk benodigde programmeerblokken (zoals een ALS-DAN blok) maar geef aan dat in een volgende les hier uitgebreider aandacht aan wordt besteed.

Ad 4. evalueren
Aan het einde van de les worden de verschillende creaties van de leerlingen aan elkaar getoond en worden de ervaringen met elkaar uitgewisseld. Doel van de evaluatie is om leerlingen van elkaars werk te laten genieten en om ervaringen met de programmeeromgeving met elkaar uit te wisselen.

7.2 de programmeerles
Voor de programmeerles gaan we uit van de structuur zoals weergegeven in Figuur 15.

Figuur 15: Onderwijsmodel voor programmeerles



Dit model bestaat uit
- een introductie,
- een unplugged activiteit,
- de fase van het gebruiken, aanpassen en creëren
- evalueren.

Dit lesmodel kan meerdere lessen beslaan en hoeft dus niet allemaal in één les aan bod komen. We zullen elk onderdeel van de les toelichten.

7.2.1 Introductie

Ter voorbereiding van de les(sen) worden tenminste drie lesdoelen geformuleerd. Een lesdoel is gericht op het vak waarin de programmeerles wordt geïntegreerd, een lesdoel is gericht op de ondersteunende computational thinking denksvaardigheid en een lesdoel is gericht op het (de) programmeerconcept(en) dat (die) centraal staat(n).
In de introductiefase wordt de context van de programmeeractiviteit duidelijk gemaakt. De context kan een thema zijn, een specifieke opdracht of eentje waarin de relatie met het vak helder wordt.

7.2.2 Unplugged activiteit

Met de unplugged acitiveit wordt het programmeerconcept helder gemaakt of wordt een CT denkvaardigheid verduidelijkt. In de speelles hebben leerlingen kennis kunnen maken met de computer en programmeren door een unplugged activiteit. Deze zogenaamde outreach unplugged activiteiten zijn gericht op de ideeën van de computer en programmeren. In de programmeerles zelf worden unplugged activiteiten ingezet om programmeerconcepten aan te leren. Voor concrete ideeën zie paragraaf 9.

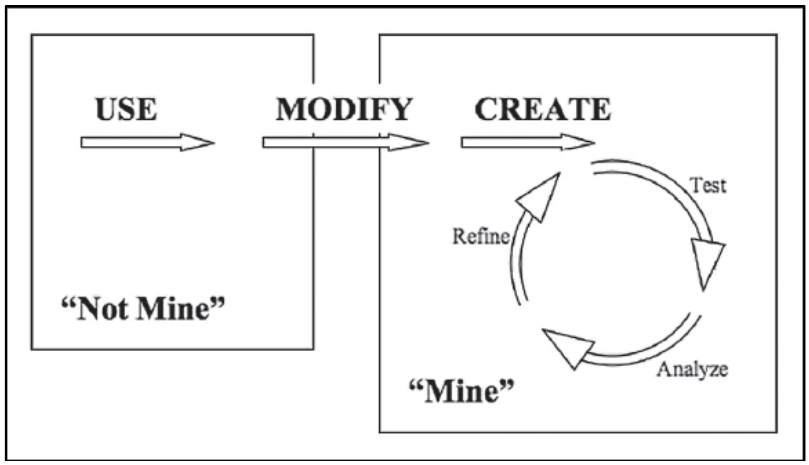
7.2.3 de fase van het gebruiken, aanpassen en creëren

In de programmeerles kiezen we voor een aanpak waarin de leerlingen geleidelijk aan steeds meer inbreng krijgen in de programmeeropdracht die ze krijgen voorgelegd en waarbij ruimte is voor CT-activiteiten. Hiermee sluiten we aan bij het uitgangspunt dat leerlingen het meeste leren als ze een duidelijke instructie krijgen waarbij ruimte is voor geleide ontdekkingen in plaats van alleen maar ontdekken en klooien (Grover et al., 2015 in Waite; 2017). Waite (2017) schrijft over deze verschuiving in de didactische aanpak rondom programmeren:

> *"The work of Papert runs as a thread throughout the curriculum frameworks devised by the various research communities. Reference is made to learners constructing knowledge as they explore and develop a personal understanding of newly introduced concepts or devices (Papert, 1980). However, balanced with this, is a call for guided instruction to ensure that learners circumnavigate a carefully constructed progression to develop a complete mental model (Garneli, Giannakos, & Chorianopoulos, 2015; Grover et al., 2015; Lye & Koh, 2014; Meerbaum-Salant et al., 2013; Schulte, 2008). Grover et al. (2015) suggest that to foster deep learning a combination of guided discovery and instruction rather than pure discovery and 'tinkering' would be more successful."*

Een didactische aanpak die aansluit bij dit uitgangspunt is het Use, Modify and Create model van Lee et al. (2011) waarin leerlingen in eerste instantie een bestaande programmacode gebruiken, deze bestaande code analyseren (Use) en aanpassen (Modify). En op basis van de opgedane kennis deze toepassen in een nieuw vergelijkbaar probleem (Create) (zie Figuur 16). Deze aanpak is specifiek geschikt voor een blokgebaseerde programeeromgeving zoals Scratch of MakeCode. Een didactische aanpak voor tekstgebasserde programmeertalen is samen te vatten in het acroniem PRIMM: Predict, Run, Investigate, Modify, Make (Sentance, Waite & Kallia; 2019). Leerlingen bestuderen in beide gevallen dus eerst de bestaande code, onderzoeken die en gaan dan pas hun eigen programmacode schrijven. Dit sluit goed aan bij de uitgangspunten van Lister (2016).

Figuur 16: Use-Modify-Crerate Model van Lee et al. (2011)



In de gebruikfase wordt klassikaal, in tweetallen of individueel een gereedstaande programmeeropdracht uitgevoerd. Dat kan bijvoorbeeld een presentatie zijn, een spel of installatie. Nadat het programma is uitgevoerd bekijken de leerlingen de onderliggende code en wordt door middel van gerichte vragen/ opdrachten de code bestudeerd. Dit wordt klassikaal besproken. In de fase van het aanpassen leren de leerlingen de code nog beter te doorgronden door aanpassingen te maken in de code en te kijken wat daarvan het effect is. Ook deze aanpassingen worden weer klassikaal besproken. In de fase van creëren krijgen de leerlingen een opdracht die ervoor zorgt dat ze het geleerde toepassen in een nieuwe situatie. Dat kan betekenen dat het bestaande project verder wordt uitgebreid waarbij ze de nieuwe kennis moeten inzetten of dat ze een nieuw project maken.

7.2.4 evaluatie

Aan het eind van de lessen wordt teruggekeken op de doelen van de lessen en worden lessen voor een volgende keer besproken.

8 Een voorbeeld

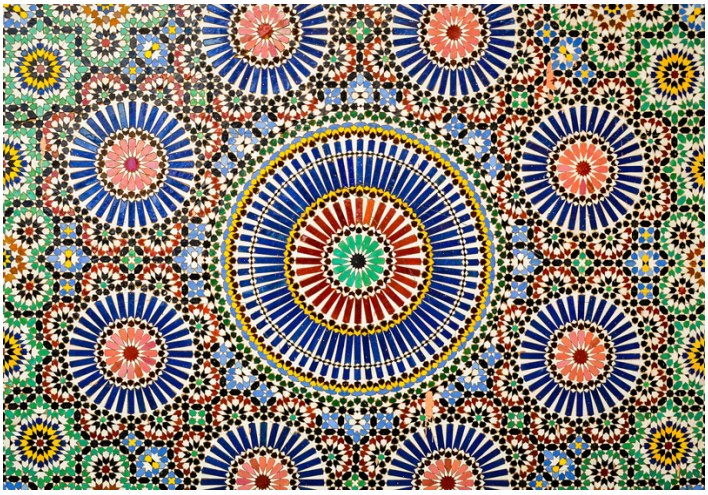
Het volgende voorbeeld doorloopt de verschillende fasen van het model. Het voorbeeld is afgeleid van de lessen zoals ontwikkeld door het UCL Knowledge Lab (2018). De volgende lesdoelen staan daarbij centraal:

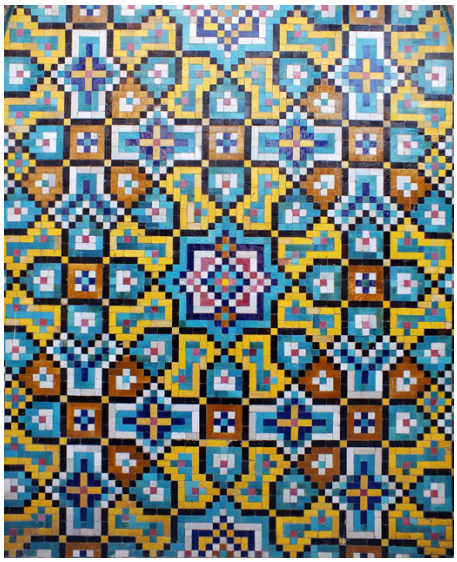
- Vak: Leerlingen construeren een patroon op basis van een aantal eenvoudige regels (Rekenen-Wiskunde)
- Compuational Thinking: Leerlingen evalueren en debuggen de code.
- Programmeerconcept: Leerlingen gebruiken functioneel in Scratch het herhalingsblok.

8.1 Introductie

Ter introductie krijgen de leerlingen de volgende afbeeldingen te zien (figuur 17)

Figuur 17: Afbeeldingen van islamitische kunst





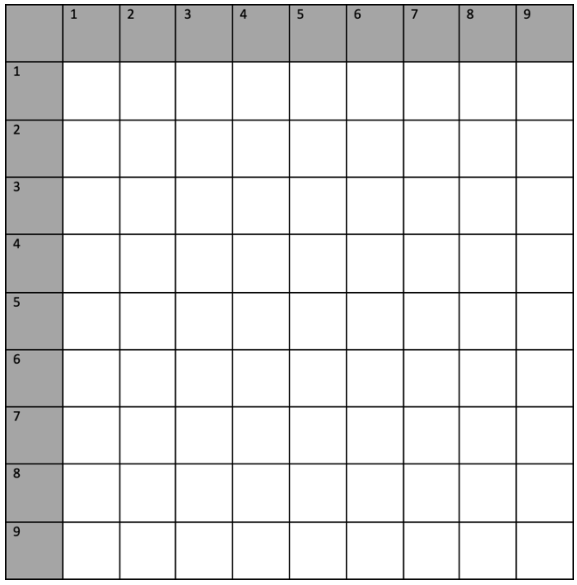
De leerkracht vertelt daarbij het volgende: In de islamitische wereld wordt in kunstwerken veel geometrische vormen gebruikt. Mensen worden niet afgebeeld zodat je die niet kunt vereren. Als je goed kijkt naar de kunstwerken zie je dat vormen steeds herhaald worden. Zo kun je met eenvoudige figuren mooie kunstwerken maken. Welke vormen zie je in dit kunstwerk veel terugkomen?

8.2 Unplugged activiteit

De leerkracht vertelt: Een belangrijk onderdeel van de islamitische kunst is het herhalen van eenvoudige vormen. Herhalen is ook een programmeerconpcept. Dit noem je ook wel een lus of loop. Jullie krijgen van mij een korte tekenopdracht waarbij je een eenvoudig patroon gaat maken.

Je volgt de aanwijzingen die ik je zo geef zo precies mogelijk op.

De leerlingen krijgen een ruitjesblad waarna ze de volgende opdracht moeten uitvoeren.



	1	2	3	4	5	6	7	8	9
1									
2									
3									
4									
5									
6									
7									
8									
9									

Ga naar de cel op de kruising van kolom 5 en rij 7.

Teken een driehoekje dat precies past in de cel.
Ga twee vakjes omhoog
Teken een driehoekje
Draai het blad een kwartslag naar links.
Ga twee vakjes omhoog

Teken een driehoekje dat precies past in de cel.
Ga twee vakjes omhoog
Teken een driehoekje
Draai het blad een kwartslag naar links.
Ga twee vakjes omhoog

Teken een driehoekje dat precies past in de cel.
Ga twee vakjes omhoog
Teken een driehoekje
Draai het blad een kwartslag naar links.
Ga twee vakjes omhoog

Teken een driehoekje dat precies past in de cel.
Ga twee vakjes omhoog
Teken een driehoekje
Draai het blad een kwartslag naar links.
Ga twee vakjes omhoog

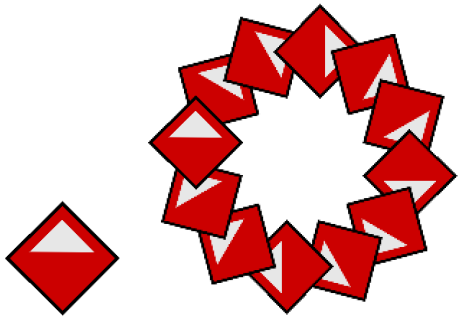
De leerkracht vraagt aan de studenten:

- Waar zit de herhaling in dit algoritme?
- Wat voor soort opdracht had ik in plaats hiervan ook kunnen geven?

8.3 Gebruiken

De leerlingen maken het Scratchproject dat op deze pagina te vinden is: https://scratch.mit.edu/projects/301557602/ en krijgen de opdracht om figuur 18 te maken.

Figuur 18: Beginfiguur en eindfiguur



De leerkracht vraagt daarna hoeveel keer ze op de sprite moeten klikken om helemaal rond te komen. Daarna vraagt de leerkracht om deze code te bekijken (zie Figuur 19).

Figuur 19. Code van de sprite

De leerkracht legt uit dat het plaatje waarop je klikt een sprite heet (zie Figuur 20).

Figuur 20. De sprite

De sprite is al geprogrammeerd om iets te doen. De leerkracht wijst de code aan en zegt: Het bovenste gele blok zorgt ervoor dat als je op de sprite klikt er iets gaat gebeuren. Daaronder staan twee blauwe blokken. Het eerste blauwe blok zorgt ervoor dat het plaatje verspringt met veertig stappen.
De leerkracht vraagt: Je moest van mij net een tekening maken. Op welke opdracht lijkt dit eerste blauwe blokje?
De leerkracht zegt: Het tweede blauwe blokje zorgt ervoor dat het plaatje draait. Op welke opdracht die je net van mij gekregen hebt, lijkt dit tweede blok?
De leerkracht zegt: Onderaan de code staat een groen blok. Stempel, heet dat blok. Daardoor stempelt het plaatje zichzelf op het doek. Net zoals je dat bij de kleuters wel eens hebt gedaan. Op welke opdracht die je net gedaan hebt lijkt dit Stempel-blok?

8.4 Aanpassen

De leerkracht vraagt de leerlingen om de bestaande code (zie Figuur 21) aan te passen op verschillende manieren.

- Verander het aantal graden dat je gaat draaien.
- Verander de grootte van de stappen.
- Zet het herhaalblok om de bestaande code zodat stappen, graden en stempel er binnen vallen.
- Probeer verschillende aantallen herhalingen uit.
- Bij hoeveel herhalingen is het plaatje helemaal rond?

Figuur 21

8.5 Creëren

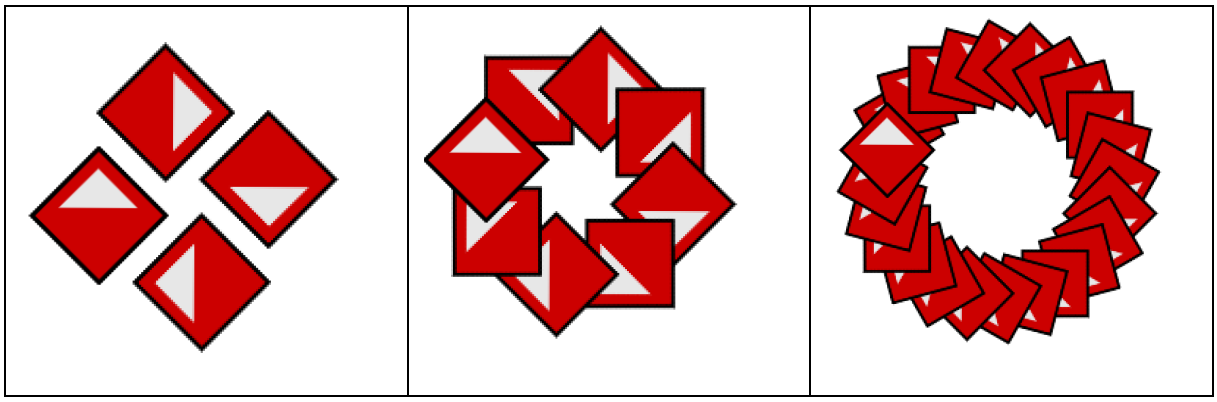
De leerlingen krijgen de volgende opdracht:

- Maak een eenvoudig patroon op papier.
- Bedenk het algoritme dat bij dit patroon hoort. Gebruik hierbij de woorden: neem … stappen en draai … graden.
- Bekijk het patroon van een ander groepje en probeer te raden wat hun algoritme is.
- Programmeer het patroon in Scratch.

8. 6 Evaluatie

In de evaluatiefase van de les kijk je terug op de les. Dit is de plek waar leerlingen het werk van elkaar kunnen bekijken en waarin je terugkomt op het programmeerconcept dat je hebt behandeld. In dit lesvoorbeeld is dat herhaling. Laat een aantal figuren (zie Figuur 22) zien en vraag welk getal in het herhalingsblok.

Figuur 22. Verschilende soorten herhalingen

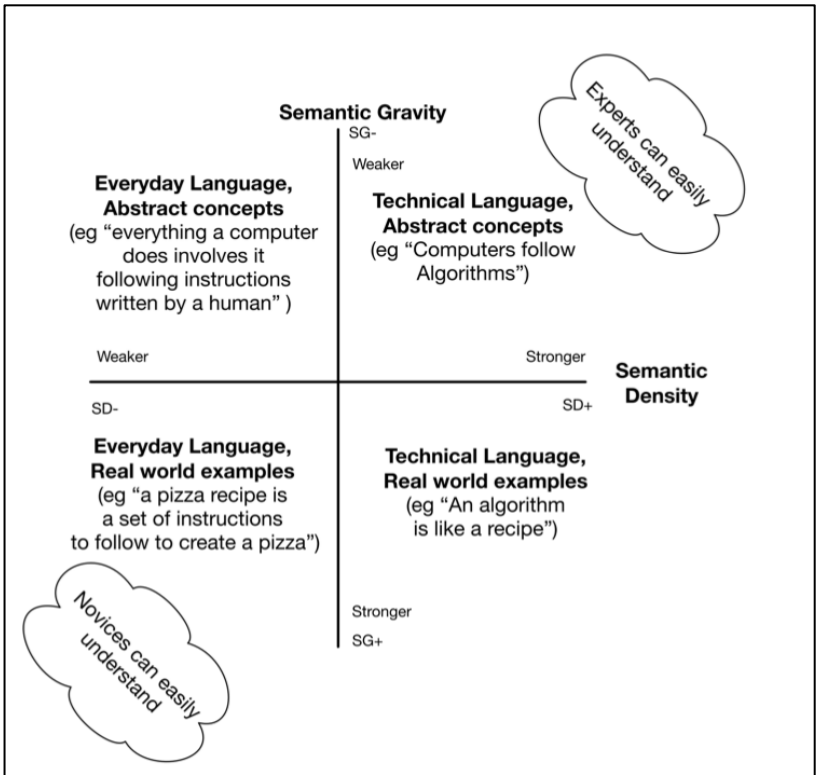


9. Unplugged activiteiten

9.1 semantic wave

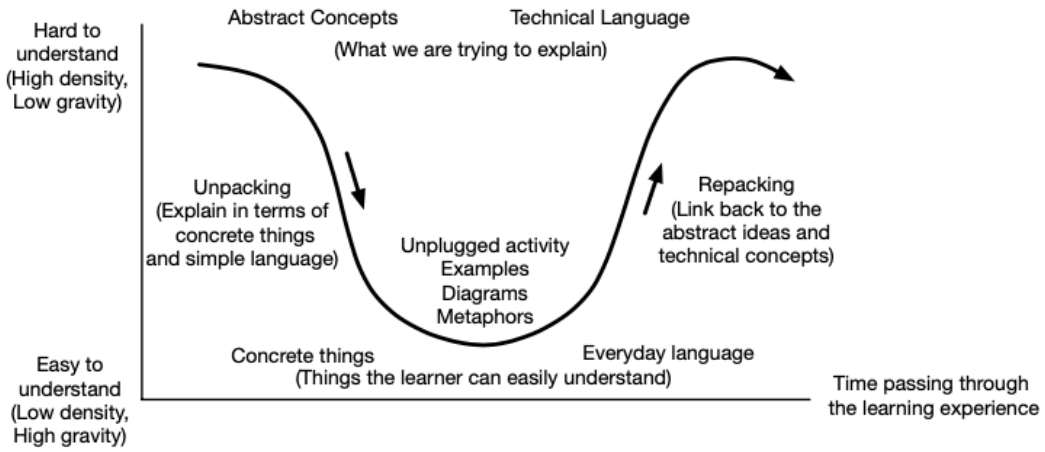
In paragraaf 7.1 stonden een paar voorbeelden van unplugged activiteiten die het idee van de computer en van programmeren introduceerden. Als introductieactiviteiten zijn die prima om uit te voeren. Maar leerlingen leren daarmee nog niet iets specifieks over een programmeerconcept. Daarvoor moet je gerichte activiteiten inzetten die dicht tegen het programmeerconcept zelf aanliggen. We laten zo per programmeerconcept een voorbeeld zien van een unplugged activiteit. Maar eerst introduceren we de zogenaamde semantic wave. Waite, Maton, Curzon en Tuttiett (2019) geven aan dat een unplugged activiteit pas effectief is als je de semantic wave doorloopt. De semantic wave hangt samen met het kwadrant dat je hieronder ziet (Figuur 23). Dit kwadrant geeft aan dat je op verschillende manieren over programmeren kunt praten. Linksonder in het kwadrant gebruik je alleen alledaagse taal met alledaagse voorbeelden (bijvoorbeeld zeggen dat je voor het maken van een pizza instructies moet volgen). Linksboven gebruik je alledaagse taal in combinatie met abstracte concepten (bijvoorbeeld: voor alles wat een computer doet, heeft het instructies nodig die geschreven zijn door een mens). Rechtsonder praat je in technische termen over een alledaags onderwerp (bijvoorbeeld: een algoritme is een soort recept). En rechtsboven praat je in technische termen over abstracte concepten (bijvoorbeeld: computers volgen een algoritme op). Van linksonder naar rechtsboven neemt de abstractie toe zou je kunnen zeggen. In termen van het kwadrant wordt dit semantic gravity and semantic density genoemd. Hoe hoger de semantic grafity and semantic density hoe abstracter de taal is die je gebruikt.

Figuur 23. Semantic Density and Semantic Gravity



Belangrijkste punt om hier uit mee te nemen is dat je op verschillende abstractieniveaus kunt praten over programmeren. En dat het voor een beginner eenvoudiger is om op een laag abstractieniveau over programmeren te kunnen praten. Maar, geven de onderzoekers aan, op dat lage abstractieniveau moet je niet alleen blijven. Daar komt de semantic wave om de hoek kijken (zie Figuur 24). In een les moet je op beide niveaus (het abstracte en alledaagse niveau) over een programmeerconcept praten en er mee aan de slag gaan. Ze noemen dat ook wel het uitpakken en weer inpakken van het programmeerconcept. In figuur XXX zie je die golfbeweging van abstracte concepten naar unplugged activiteiten terug naar abstracte concepten.

Figuur 24. Semantic wave in een les



Wat betekent dit nu concreet? Als je een unplugged activiteit doet, zorg dan dat leerlingen eerst weten met wat voor soort programmeerconcept ze straks aan de slag gaan. Daarna volgt de unplugged activiteit die het programmeerconcept op een toegankelijke manier uitlegt. Tot slot moet de unplugged activiteit weer teruggekoppeld worden naar het abstracte programmeerconcept.

9.2 Voorbeelden unplugged activiteiten

Per programmeerconcept volgt nu een voorbeeld van een unplugged activiteit. Bedenk dat bij elk voorbeeld de semantic wave nog moet worden gemaakt.

Sequentie

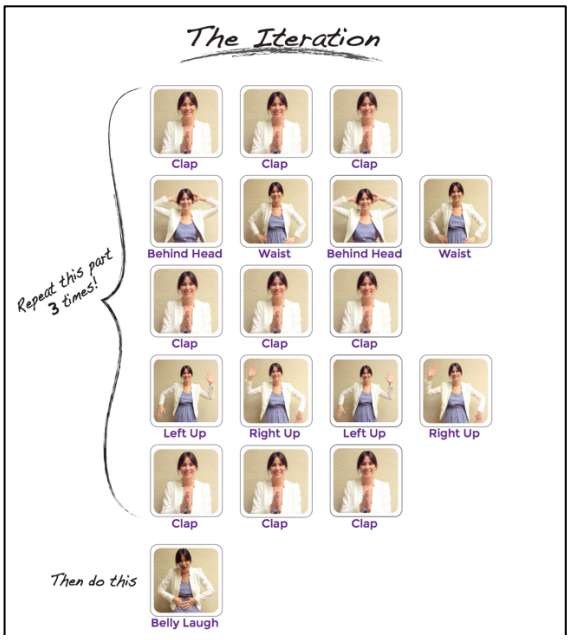
Het spel Robot Turtle (zie Figuur 25) maakt leerlingen duidelijk dat een opdracht in een specifieke volgorde uitgevoerd moet worden. Alleen dan kunnen de schildpadden de diamanten verzamelen. De hagelslagrobot uit de speelles (paragraaf 7.1) is ook een mooi voorbeeld van een sequentieopdracht.

Figuur 25. Robot Turtle spel

Herhaling

Met het spel Getting Loopy van code.org (https://code.org/curriculum/course1/12/Teacher) ga je een soort opvoering maken (zie Figuur 26). Alleen kan de instructie nog wel verkort worden. Er zit namelijk nogal wat herhaling in. Ontdek met de leerlingen samen wat steeds terugkomt en herhaalt kan worden.

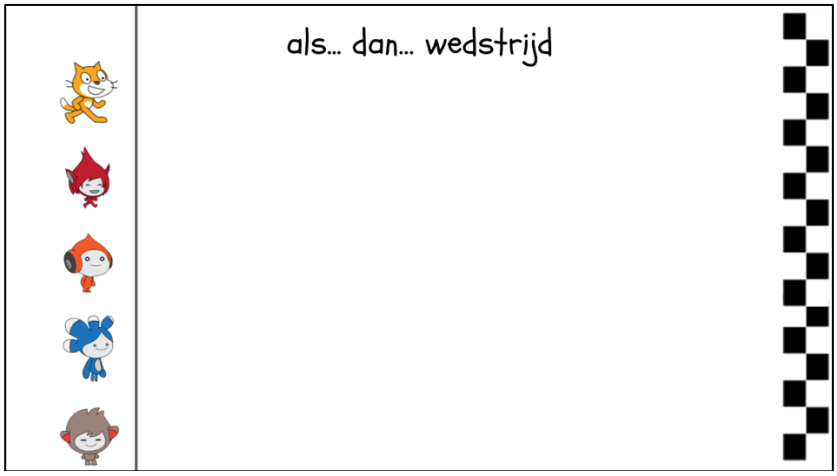
Figuur 26. Werkblad bij het spel Getting Loopy



Conditie

Een conditie is een als-dan(-anders) uitspraak. Je kunt een wedstrijd houden waarbij leerlingen een stap naar voren mogen nemen als ze aan een bepaalde voorwaarde voldoen (zie Figuur 27). Een voorbeeld vind je op: https://makecode.microbit.org/courses/csintro/conditionals/unplugged.

Figuur 27. Opstelling bij een ALS-DAN-wedstrijd



Voorbeelden die je kunt gebruiken:

Ronde 1: Als dan

- Als je bruin haar hebt neem dan één stap naar voren.
- Als je blauwe ogen hebt neem dan één stap naar voren.
- Als je een broek aan hebt neem dan één stap naar voren.

Ronde 2: Als dan anders

- Als je naam begint met een medeklinker neem dan één stap naar voren anders neem je twee stappen naar voren.
- Als je je propedeuse hebt gehaald neem je twee stappen naar voren anders neem je één stap naar voren.
- Als je vooraan staat neem je nul stappen naar voren anders neem je twee stappen naar voren.

Ronde 3: Als ... EN als... dan.... anders

- Als je een jongen bent EN je hebt blond haar dan doe je twee stappen naar voren anders neem je één stap naar voren.
- Als je een meisje bent EN je hebt een armband om dan doe je twee stappen naar voren anders neem je één stap naar voren.
- Als je deze maand jarig bent EN je ouder bent dan 19 neem je twee stappen naar voren anders neem je één stap naar voren.

Ronde 4: Zelf statements bedenken door de leerlingen

- **...**

Variabele

Variabelen zorgen ervoor dat de computer een bepaalde waarde onthoudt in het programma. Dat kan een getal zijn maar ook een woord. Om de werking van een variabele duidelijk te maken kun je een wedstrijdje doen waarbij leerlingen een score moeten bijhouden op een wisbordje. Elke keer als ze een punt hebben gescoord, wordt de huidige waarde uitgewist en de nieuwe waarde opgeschreven. Dit lesidee kun je ook terugvinden op https://www.barefootcomputing.org/resources/variables-unplugged-activity. Een ander voorbeeld van een unplugged activieit over variabelen vind je op de website van London Computing (https://teachinglondoncomputing.org/resources/inspiring-unplugged-classroom-activities/the-box-variable-activity/) daarbij wordt een doos-metafoor gebruikt.

Maar wel een speciale doos waarin een shredder en fotokopieerder zit. Een variant hierop is dezelfde activiteit maar dan met een wasknijper. De wasknijkper kan maar 1 blaadje (de waarde) bevatten. Als er een nieuw blaadje in moet, valt het vorige blaadje eruit. Een derde unplugged activiteit over variabelen is de enveloppen activiteit. Hierbij ligt de nadruk op het feit dat een een variable een soort placeholder is. De uitleg van de activiteit kun je hier vinden: https://studio.code.org/s/course4/lessons/4/levels/1.

Lijst

Een lijst (of array) is een verzameling van waarden die je opslaat. Die analogie kun je ook gebruiken om een unplugged activiteit vorm te geven. Op de website https://minecraft.makecode.com/courses/csintro/arrays vind je een mooi voorbeeld als het gaat over het bevragen van kinderen naar spullen die ze verzamelen. Hoeveel spullen zitten er in hun verzameling? Wat voor soort spullen zijn het? Hoe hebben ze de spullen geordend? Hoe hebben ze ervoor gezorgd dat ze hun spullen gemakkelijk terug kunnen vinden? Een vervolgactiviteit vind je op https://teachinglondoncomputing.org/the-bubblesort-activity/ waarin het gaat over het op orde brengen van een lijst.

10 Soorten vragen en opdrachten

In je les stel je vragen en geef je opdrachten. In deze paragraaf krijg je handvatten om vragen te stellen en opdrachten te geven.

10.1 Vragen op basis van het block-model

In de Gebruiken (en begrijpen) fase van de les stel je leerlingen vragen om hen te helpen de code te doorgronden. Je kunt verschillende soorten vragen stellen afhankelijk van de ervaring van de leerlingen. Voor de indeling van deze vragen kun je gebruik maken van het block-model van Schulte (2008). Het block-model is bedoeld als een leerweg in het steeds beter begrijpen van de code. Het block-model vind je in Figuur 28.

Figuur 28: block-model van Schulte.

Macro structure	(Understanding the) overall structure of the program text	Understanding the „algorithm“ of the program	Understanding the goal/ the purpose of the program (in its context)
Relations	References between blocks, e.g.: method calls, object creation, accessingdata...	Sequence of method calls „object sequence diagrams	Understanding how subgoals are related to goals, how function is achieved by subfunctions
Blocks	'Regions of Interests' (ROI) that syntactically or semantically build a unit	Operation of a block, a method, or a ROI (as sequence of statements)	Function of a block, maybe seen as sub-goal
Atoms	Language elements	Operation of a statement	Function of a statement. Goal only understandable in context
	Text surface	**Program execution (data flow and control flow)**	**Function (as means or as purpose), goals of the program**
	Structure		**Funtion**

Het block-model kun je, als een muurtje van onder naar boven, als volgt lezen. Van klein naar groot zijn er vier niveaus: atom (atomen), blocks (blokken), relations (relaties) en macro-structure (macro-structuur). Bij die vier niveaus kun je het hebben over de structure (structuur) en de function (functie). Met structuur wordt de opbouw van het programma bedoeld en met functie wat je er mee wilt bereiken. Oftewel: hoe zit het programma in elkaar (structuur) en wat doet het programma (functie) Als leerlingen kennis maken met een nieuw programmeerprobleem dan kun je gebruik maken van die vier niveaus en van structuur en functie.

Voor de programmeerles die je moet geven kun je gebruik maken van dit block-model door de vragen die je stelt te relateren aan het block-model. Dat wil zeggen: vragen stellen die gericht zijn op het atoom-niveau (een enkele programmeerregel of programmeerblokje), blok-niveau (een samenhangend stukje code of samenhangende programmeerblokken), relaties (relatie tussen meerdere samenhangende stukjes code of samenhangende programmeerblokken) of macro-strucuur (over de code als geheel).

Een voorbeeld. In de les over het programmeren van een mozaïek heb je kennis gemaakt met de code in figuur 29.

Figuur 29: code van de mozaïekles

Laten we kijken wat voor vragen we kunnen stellen als we gebruik maken van het block-model in de fase van gebruiken. Tabel 4 houdt de volgorde van het block-model van Schulte aan. Het moet dan ook van beneden naar boven gelezen worden.

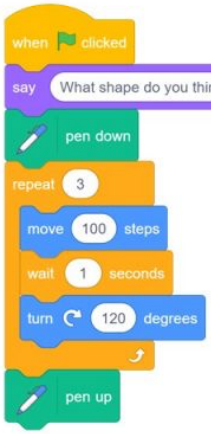
Tabel 4: Vragen op basis van het block-model

Vragen op het macro-structuurniveau	Kun je de code van het programma uitleggen in je eigen woorden?
Vragen op het relatie-niveau	Er staan twee programma's in beeld. Hoe werken die twee programma's samen?
Vragen op het block-niveau	Kun je uitleggen hoe de programmeerblokken tussen het herhaalblok samenwerken?
Vragen op het atoom-niveau	Welk programmeerblok zorgt ervoor dat de sprite steeds een stukje verspringt? Welk programmeerblok zorgt ervoor dat de sprite steeds een stukje draait? Welk programmeerblok zorgt ervoor dat er steeds een sprite bij komt? Welk programmeerblok zorgt ervoor dat alles herhaald wordt? Welk programmeerblok zorgt ervoor dat het programma start als je op de sprite klikt?

10.2 Opdrachten van Baggy

Je kunt in de Modify (Aanpassen-fase) verschillende opdrachten geven om de code aan te passen. Phil Bagge geeft op zijn website (http://code-it.co.uk/codecomp/) een mooie opbouw hiervoor. Die aanpassingen beginnen klein en worden steeds groter. Zie figuur 30.

Figuur 18: screenshot van Modify-opdrachten van Baggy.



Stages of Code Modification	Examples
Minor changes non operational	Changing text in the say block Changing the seconds in the say block
Minor operational changes	Changing the start block type Changing the amount of time the program waits within the loop Changing the move x steps to change the length of each side
Major changes non operational	A new sprite A new background Changing the loop into a sequence Changing the code into a procedure
Major changes operational	Making more than one change towards a greater purpose such as adapting it to draw a square Changing the number of repeats Changing the angle

Toegepast op het voorbeeld van de mozaïekopdracht zou dat er bijvoorbeeld uit kunnen zien zoals in tabel 5.

Tabel 5: Voorbeeldvragen op basis van Baggy.

Fasen van code aanpassen	**Voorbeelden**
Kleine veranderingen – niet operationeel	Pas de grootte van de stappen aan.
Kleine operationele veranderingen	Pa de volgorde waarin de blokken tussen het herhaalblok staan aan
Grote veranderingen – niet operationeel	Maak hetzelfde figuur met een nieuwe sprite.
Grote operationele veranderingen	Verander het aantal herhalingen.

Over de vragen

Het is niet zo dat je per se uit elke categorie een vraag moet stellen of een opdracht moet geven. De vragen en opdrachten kunnen je helpen om kinderen de code beter te laten begrijpen. Als je merkt dat de vraag die je stelt tijdens de les te ingewikkeld is of juist te eenvoudig kun je schakelen tussen de verschillende soorten vragen en opdrachten.

Uitgangspunt is iedereen heeft instructie nodig. Kinderen die het snel oppikken en kinderen die er meer moeite mee hebben. Die instructie vindt plaats tijdens de unplugged activiteit en het vervolg/ reflectie daarop. De vragen die je stelt zijn mede afhankelijk van hoe snel kinderen het oppikken. Kinderen die meer moeite met de opdracht hebben, help je op weg door hen vragen te stellen die meer afgebakend zijn. Dus vragen op atoom en block niveau. Kinderen die dit sneller oppikken kun je vragen stellen op relatie en marco-structuur niveau. Je zou drie groepen kunnen hanteren. Een groep voor de versnelde instructie die met macro-structuur en relatie vragen aan de slag kunnen. Een groep voor de gewone instructie

die met aantal atoom en block vragen aan de slag kunnen. En een groep voor de verlengde instructie die aan de hand genomen moet worden en waarmee je samen de code nabouwt en samen de code aanpast en samen de nieuwe opdracht doorloopt.

11 Verder dan 1 les

In paragraaf 12 hebben we de opzet van 1 les beschreven. Uitgangspunt hierbij was dat leerlingen nog niet eerder geprogrammeerd hebben en het na deze les ook niet meer zullen gaan doen. Een onafhankelijke les dus midden in het onderwijs. Maar wat als je meer dan 1 programmeerles wilt geven? Hoe kun je dat dan aanpakken? In deze paragraaf krijg je handvatten om een lessenserie te ontwerpen rondom programmeren. Uitgangspunt hierbij is het werken met een blokgebaseerde programmeeromgeving zoals Scratch waarbij leerlingen steeds meer zelf moeten uitzoeken.

11.1 Koppeling met vakgebied en vakoverstijgende doelen

Als je meerdere programmeerlessen gaat verzorgen dan zijn het *philosophical argument* en *problem solving argument* zoals Honegger (2015) die benoemd mooie uitgangspunten. Het *philosophical argument* zegt dat programmeren een manier is om de wereld om ons heen te begrijpen en mede vorm te geven. Je leert nieuwe belangrijke concepten kennen die je inzicht geven in de wereld om je heen. Met *problem solving argument* geeft Honneger aan dat programmeren nog een context is waarin je kunt oefenen met het oplossen van problemen. Programmeren om de wereld om je heen beter te begrijpen en vorm te geven kun je koppelen aan verschillende vakgebieden op de basisschool. In Tabel 6 een paar voorbeelden vanuit de kerndoelen. Programmeren is een verrijking op het gebied van de mogelijkheden van deze vakken. Over het *problem solving argument* meer in de volgende paragraaf.

Tabel 6: Ideeën voor lessenseries

Vak	**kerndoelen**	**Idee voor lessenserie**
Nederlands	De leerlingen leren naar inhoud en vorm teksten te schrijven met verschillende functies, zoals: informeren, instrueren, overtuigen of plezier verschaffen.	Schrijven van interactieve verhalen
Rekenen/wiskunde	De leerlingen leren aanpakken bij het oplossen van rekenwiskundeproblemen te onderbouwen en leren oplossingen te beoordelen.	Programma's maken waarmee rekenkundige/ wiskundige problemen opgelost kunnen worden.
Natuur en techniek	De leerlingen leren onderzoek doen aan materialen en natuurkundige verschijnselen, zoals licht, geluid, elektriciteit, kracht, magnetisme en temperatuur. De leerlingen leren oplossingen voor technische problemen te ontwerpen, deze uit te voeren en te evalueren.	Apparaten programmeren om specifiek probleem op te lossen.

Kunstzinnige oriëntatie	De leerlingen leren beelden, muziek, taal, spel en beweging te gebruiken, om er gevoelens en ervaringen mee uit te drukken en om er mee te communiceren.	Animaties Muziek Games

11.2 Aanleren programmeerconcepten

Uitgangspunt is het systematisch leren programmeren. Onderdeel daarvan is het gebruik van de verschillende programmeerconcepten. In paragraaf 10 noemden we de volgende programmeerconcepten: sequentie, herhaling, selectie, variabele en lijst (of rij of reeks). Deze programmeerconcepten zijn de gereedschappen waarmee je verschillende programma's kunt maken. Combineer je deze programmeerconcepten met elkaar dan kun je er complexere programma's mee maken. Afhankelijk van de doelen die je stelt, heb je een of meerdere van deze programmeerconcepten nodig. In verschillende lessen kun je leerlingen deze programmeerconcepten aanleren. Per les is het handig om een 1 programmeerconcept centraal te stellen. Als de verschillende programmeerconcepten aan bod zijn gekomen dan kun je ze met elkaar combineren. Afhankelijk van de doelgroep kan dit sneller of moet dit langzamer. Deze snelheid zit vooral in de type opdrachten/ ondersteuning die je moet geven. In het begin zullen meer opdrachten gericht zijn op het atoom niveau. Aan het eind van de lessenserie zullen opdrachten meer gericht kunnen zijn op de macrostructuur.
Naast het combineren van programmeerconcepten kunnen de individuele programmeerconcepten ook nog weer op verschillende niveaus aangeboden worden. Op: http://everydaycomputing.org/public/visualization/ vind je overzichten van verschillende programmeerconcepten en het leertraject dat je voor zo'n programmeerconcept kunt doorlopen.

11.3 Leren werken met ontwerpeisen

Beheersen de leerlingen alle programmeerconcepten die ze nodig hebben in jouw lessenserie? Dan kun je meer de nadruk leggen op de ontwerpeisen. Uitgaan van ontwerpeisen vraagt van leerlingen *problem solving* vaardigheden. De ontwerpeisen moeten door leerlingen omgezet kunnen worden in een algoritme en dat moet weer omgezet kunnen worden in bijbehorende code. Leerlingen werken daarmee dus ook aan hun computational thinking vaardigheden.
Werken met ontwerpeisen komen niet pas aan het einde van de lessenserie aan bod. Gedurende de hele lessenserie vraag je namelijk al aan leerlingen in de fase van creëren om op basis van ontwerpeisen een nieuwe opdracht te maken en dus een probleem op te lossen. Maar het karakter van de lessen kan wel veranderen gedurende de lessenserie. In het begin van de lessenserie is de transfer van het geleerde naar de nieuwe toepassing niet zo groot en bied je meer begeleiding op het toepassen van de programmeerconcepten. Aan het eind kan de transfer groter zijn en kan de zelfstandigheid bij het oplossen van de problemen toenemen. In paragraaf 15.4 wordt dit meer toegelicht.

11.4 Voorbeeld lessenserie

Stel je wilt een lessenserie maken waarin leerlingen leren om een verhaal na te vertellen (in het kader van Nederlands). Een verhaal heeft karakters, speelt zich ergens af, er komt actie in voor, de karakters gaan met elkaar in gesprek, en heeft een begin, midden en eind. Deze aspecten wil je terug laten komen in je lessenserie. In elke les kunnen verschillende aspecten

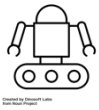
aan bod komen. In les 1 laat je karakters bijvoorbeeld met elkaar praten. In les 2 zorg je ervoor dat karakters door verschillende landschappen kunnen bewegen. In les 3 zorg je ervoor dat karakters op elkaar reageren als ze in actie komen. In deze lessen laat je de leerlingen de bijpassende programmeerconcepten oefenen. In les 1 is dat bijvoorbeeld het sequentieel programmeren. In les 2 is dat bijvoorbeeld de conditie. In les 3 is dat bijvoorbeeld herhaling. In elk van deze lessen zit een creëeropdracht waarbij ze het geleerde moeten toepassen op een nieuw (vergelijkbaar probleem). In les 4 , 5 en 6 gaan leerlingen dan aan de slag om de geleerde programmeerconcepten toe te passen op een eindopdracht waarin de ontwerpeneisen zijn gebaseerd op de geleerde programmeerconcepten. In deze laatste drie lessen is er onder meer aandacht voor het op een rijtje zetten van de ontwerpeisen en wat dit betekent voor de te maken algoritmes en welke programmeerconcepten daarvoor nodig zijn. Samen met leerlingen worden activiteiten gedaan die een beroep doen op de verschillende aspecten van computational thinking: abstractie, generalisatie/ patronen, decompositie, algoritmisch denken en debuggen/ evalueren. Het is belangrijk om dit eerst samen te doen want het zijn weer nieuwe vaardigheden die ook weer geoefend moeten worden.

12 Debuggen

In de fase van het aanpassen en creëren zal niet alles in een keer goed gaan met het programmeren. Zoals je bedacht hebt hoe het eruit moet zien of wat het moet doen lukt niet altijd in een keer. Belangrijk bij programmeren is dat je ook fouten op leert sporen. Phil Bagge (2015) zet specifiek voor Scratch (maar ook geschikt voor andere blokgebaseerde programmeertalen) de debugstrategieën op een rijtje (zie Tabel 7). In een lessenreeks hoeven niet alle debugstrategieën in een keer aan bod te komen. Je kunt er een aantal uitkiezen die je voor deze opdracht heet meest handig vindt.

Tabel 7: Debugstrategieën voor blokgebaseerde programmeertalen

Kleurverschillen	
Vormverschillen	
Hoeveelheid blokken	

Patronen en vormen	
Hardop voorlezen	
Hardop voorlezen aan ander	
Hardop voorlezen en laten meelezen	
Leerkracht lees code voluit voor	
Waarom werkt dit stukje code niet?	
Leg code uit aan medeleerling en wijs de code aan	
Wat gebeurt er met een variabele na elke herhaling?	
Verdeel en heers	
Code uitleggen aan eendje	

13 Lesideeën en Achtergrondinformatie

In paragraaf 8 heb je een lesidee aangereikt gekregen. Nu hoef je niet alle lesideeën zelf helemaal te verzinnen. Er zijn al heel veel leerkrachten en ontwikkelaars die lessen/ ideeën hebben gemaakt en/of verzameld. In dit lijstje vind je een aantal van deze plekken:

- Linken naar de ScrathED-community:
    - http://scratched.gse.harvard.edu/resources/scratch-across-every-subject-recap.html
    - http://scratched.gse.harvard.edu/resources/scratch-projects-across-curriculum
    - http://scratched.gse.harvard.edu/resources/scratch-across-every-subject-literacy
    - http://www.literacyfromscratch.org.uk/
    - http://scratched.gse.harvard.edu/resources.html
    - http://scratched.gse.harvard.edu/guide/curriculum.html
- Scratch: https://scratch.mit.edu/ideas
- MicroBit met MakeCode: https://makecode.microbit.org/#
- Website van Phil Bagge: http://code-it.co.uk/
- Overzicht op website Mediawijsheid: https://www.mediawijsheid.nl/programmeren/
- Codekinderen: https://maken.wikiwijs.nl/100525/CodeKinderen
- CS Unplugged: http://www.csunplugged.nl/
- CS Unplugged (Engels): https://csunplugged.org/en/
- https://www.ucl.ac.uk/ioe/research/projects/scratchmaths

14 Literatuurlijst

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). *A K-6 Computational Thinking Curriculum Framework: Implications for Teacher Knowledge*. Educational Technology & Society, 19 (3), 47–57.

Bagge, P. (2015). *Code It How To Teach Programming Using Scratch: Teacher's Handbook (Code-IT Primary Programming) A complete KS2 Computer Science study programme*. Londen: University of Buckingham Press

Bell T., Vahrenhold J. (2018) CS Unplugged—How Is It Used, and Does It Work?. In: Böckenhauer HJ., Komm D., Unger W. (eds) *Adventures Between Lower Bounds and Higher Altitudes. Lecture Notes in Computer Science*, vol 11011. Springer, Cham

CAS (2013). Computing in the national curriculum. A guide for primary teachers. Newnorth Print, Ltd. Bedford

Honegger, B.D. (2015). *We are all excited, but why?* Geraadpleegd op 26-8-2019 via https://beat.doebe.li/talks/scratch15/index.html

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., ... & Werner, L. (2011). *Computational thinking for youth in practice*. *Acm Inroads*, *2*(1), 32-37.

Lister, R.(2016). Toward a Developmental Epistemology of Computer Programming. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education (WiPSCE '16)*. ACM, New York, NY, USA, 5-16.

National Research Council (2010). *Report of a Workshop on the Scope and Nature of Computational Thinking. Washington, DC*: The National Academies Press.

Raad voor Cultuur (2005). *Mediawijsheid. De ontwikkeling van Nieuw Burgerschap.* Den Haag: Raad voor Cultuur.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for All. *Communications of the ACM*, vol. 52, no. 11, pp. 60-67 (Nov. 2009).

Selby, Cynthia and Woollard, John (2013) *Computational thinking: the developing definition* University of Southampton (E-prints) 6pp.

Sentance, S. Waite, J. & Kallia, M. (2019) *Teaching computer programming with PRIMM: a sociocultural perspective*. Computer Science Education, 29:2-3, 136-176,

SLO (2016). Digitale geletterdheid uitgewerkt in digitale vaardigheden. Geraadpleegd op 26-8-2019 via: http://curriculumvandetoekomst.slo.nl/21e-eeuwse-vaardigheden/digitale-geletterdheid

UCL Knowledge Lab (2018). ScratchMath. Geraadpleegd op 26-9-2019 via: https://www.ucl.ac.uk/ioe/research/projects/scratchmaths

Waite, J. (2017). Pedagogy in teaching Computer Science in schools: A Literature Review. Queen Mary University of London and King's College London

Waite, J., Maton, K., Curzon, P., & Tuttiett, L. (2019, September). Unplugged computing and semantic waves: Analysing crazy characters. In *Proceedings of the 1st UK & Ireland Computing Education Research Conference* (pp. 1-7).

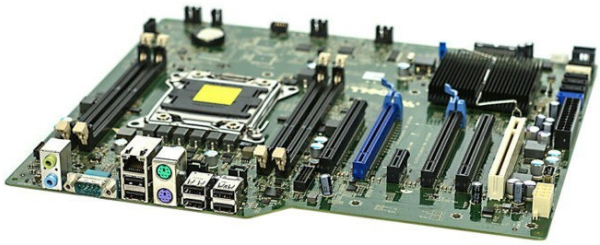
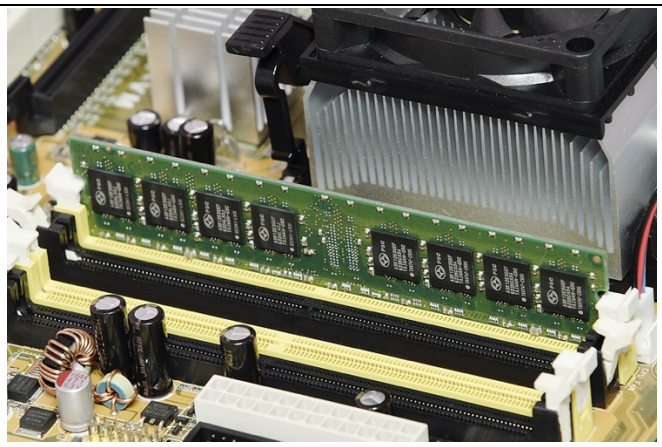
Wing, J.M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35. https://www.nap.edu/read/12840/chapter/3#13

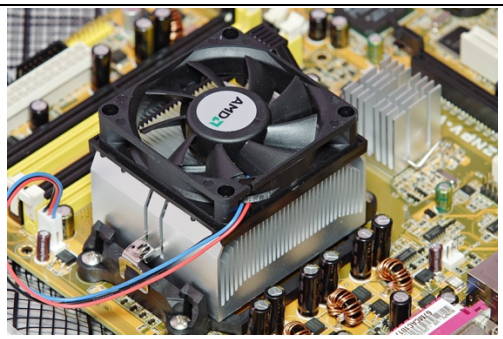
Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717-3725.

Bijlagen

Bijlage A – Kennis over de computer

Hardware

Onderdelen	Foto	Wat doet het?
Processor		Hier voert de computer de berekeningen mee uit. Hier staat het computerprogramma op. Het computerprogramma bestaat uit algoritmes die beslissen wat er met de signalen die binnenkomen moet gebeuren.
Moederbord		Printplaat waarop alle onderdelen die in de computer zitten mee verbonden zijn.
Intern geheugen		Het geheugen dat direct op het moederbord is geplaatst. Het is snel en de processor gebruikt het om instructies die daar zijn opgeslagen te lezen.

Videokaart		De videokaart zorgt voor beelden op een scherm. Bijvoorbeeld op je computerscherm. Een snelle videokaart zorgt ervoor dat beelden vloeiend worden getoond.
Harde schijf		Een extern geheugen waar informatie opgeslagen kan worden zoals bijvoorbeeld bestanden.
Voeding		Zorgt ervoor dat de computer stroom krijgt en daarmee dus aan kan gaan.
Koeling		Zorgt ervoor dat de processor het niet te heet krijgt door de vele berekeningen die het moet uitvoeren.

Naast de onderdelen die je in de computer ziet, zijn er natuurlijk nog onderdelen die je gebruikt om de computer **input** te geven.

Muis	
Toetsenbord	
Microfoon	
Webcam	

Touchpad	
Scanner	

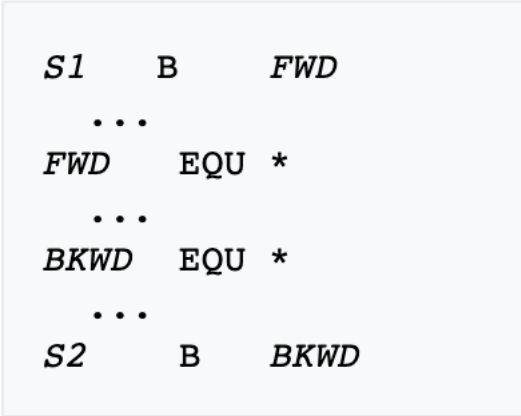
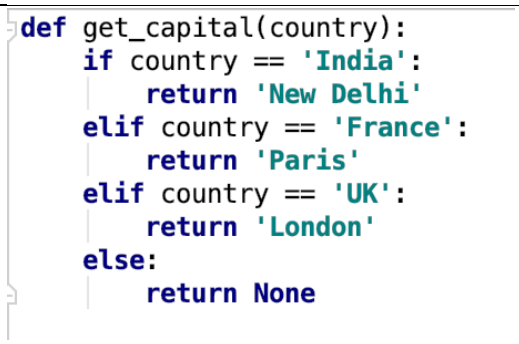
En tenslotte geeft de computer ook output.

Beeldscherm	

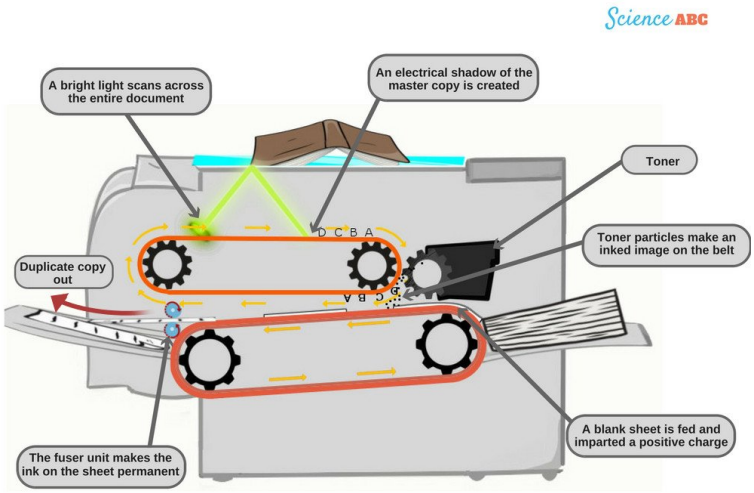
Printer	

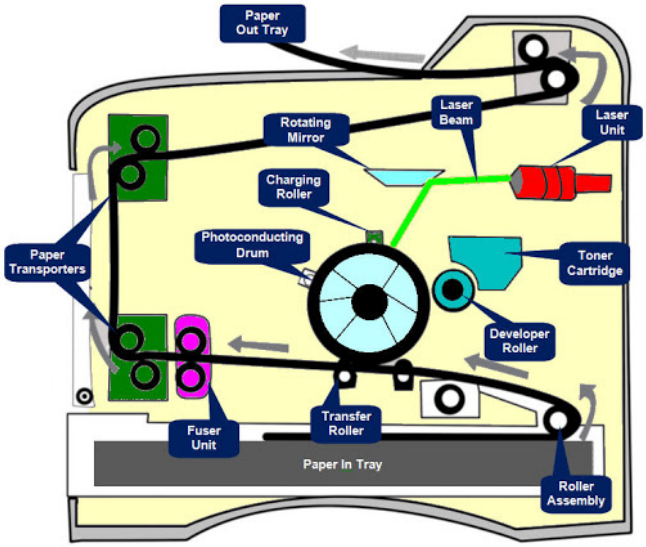
Software
Software vertelt de computer wat het moet doen. De manier waarop software wordt geprogrammeerd kun je op verschillende manieren indelen: lagere programmeertalen en hogere programmeertalen.

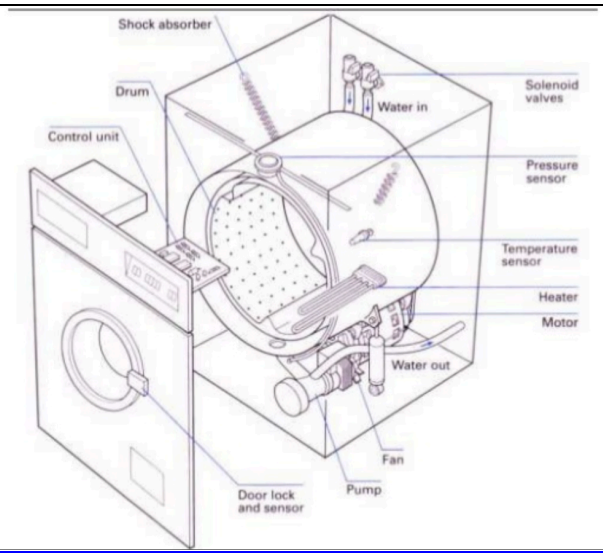
<table>
<tr><td colspan="3">Lagere programmeertaal</td></tr>
<tr><td>Machinetaal</td><td>
<table>
<tr><th>Binair</th><th>Decimaal</th></tr>
<tr><td>00000</td><td>0</td></tr>
<tr><td>00001</td><td>1</td></tr>
<tr><td>00010</td><td>2</td></tr>
<tr><td>00011</td><td>3</td></tr>
<tr><td>00100</td><td>4</td></tr>
<tr><td>00101</td><td>5</td></tr>
<tr><td>00110</td><td>6</td></tr>
<tr><td>00111</td><td>7</td></tr>
<tr><td>01000</td><td>8</td></tr>
<tr><td>01001</td><td>9</td></tr>
<tr><td>01010</td><td>10</td></tr>
</table>
</td><td>Werkt met 0 en 1. Dit is de taal die de computer begrijpt. Het zijn in wezen elektrische signalen die aan (1) of uit (0) kunnen staan. Een 0 of een 1 noem je een bit. 8 Bits achter elkaar noem je een byte. Bytes maken gebruik van transistors en het registergeheugen. Een transistor is een schakelaar die een stroom wel of niet doorlaat. Op een processor zitten miljarden van deze transistoren. In het registergeheugen, zijn alle bitrijen opgeslagen. De processor haalt de informatie op uit het geheugen en bepaald</td></tr>
</table>

		wat er mee gedaan moet worden.
Assembleertaal		Een symbolische weergave van de machinetaal. Elke regel voert 1 instructie uit. Een zogenaamde assembler zet de assembleertaal voor de computer om in machinetaal.
Hogere programmeertaal		De programmeertalen waarin bijna alle programmeurs werken is een hogere programmeertaal. De instructies in de hogere programmeertaal zijn goed voor mensen te begrijpen. Voor de computer moeten de instructies vertaald worden door een zogenaamde compiler. Die maakt er begrijpelijke instructies voor de computer van. Er zijn heel veel vershillende soorten programmeertalen. Het voorbeeld hiernaast is Python.

Bijlage B – computer in andere apparaten

Apparaat	Werking van het apparaat
Kopieerapparaat	Om een kopie van een papier te maken met een kopieerapparaat leg je het originele papier omgekeerd op een glazen plaat. Het originele papier wordt belicht. Dit licht wordt weerkaatst en komt op een elektrostatisch geladen rol terecht. Waar het origineel zwart is weerkaatst het licht niet en waar het origineel wit is weerkaatst het licht wel. Het licht dat wel terecht komt op de elektrostatische rol zorgt ervoor dat de rol op die plekken ontlaat. De rol is daardoor alleen op de plekken waar het licht niet weerkaatste nog steeds elektrostatisch geladen. De elektrostatische rol trekt daarna de inkt (toner) aan op de plekken waar de rol nog geladen is. Daarna komt papier langs de rol. Het papier wordt ook elektronisch geladen. Het papier trekt daardoor de toner naar zich toe. Daarna rolt het papier tussen twee rollers door die verwarmd worden. Die rollers smelten de toner op het papier. Zie ook: https://www.youtube.com/watch?v=JtA7B_zSrjE **Inkjetprinter**  De hierboven beschreven manier van werken hoort bij analoge kopieerapparaten. Moderne kopieerapparaten zijn een combinatie van een scanner en een laserprinter. Een scanner maakt van het originele papier een digitale afbeelding. De digitale afbeelding wordt gebruikt om een laser aan te sturen die op een elektrostatische rol is gericht. De laser kan heel precies aangeven welk deel van de rol ontladen moet worden en welke niet. Ook daarna hecht de inkt zich op de rol en komt vandaaruit op het papier terecht. Doordat het origineel een digitale afbeelding wordt, kan die nog worden bewerkt voordat het wordt gekopieerd. De computer in het kopieerapparaat kan het originele papier dus opslaan in het geheugen. Daarnaast kan de computer de specifieke uitvoer regelen van het origineel: hoeveelheid, kleur, grootte.

	Laserprinter 
Wasmachine	Om de was schoon te maken met behulp van de wasmachine wordt water toegevoegd, wordt het water verhit en draait de trommel voor een bepaalde tijd. Na afloop wordt het water afgevoerd. Het water wordt toegevoegd door het openen en sluiten van kleppen. Het verhitten van het water gebeurt met een verwarmingselement. De trommel draait door het laten draaien van een motor die is aangesloten op de trommel. Het water afvoeren gebeurt door het openen van de afzuigpomp. In oude modellen werd dit op een mechanische manier geregeld. In de moderne wasmachines stuurt een computer de verschillende onderdelen van het apparaat aan door het geven van elektrische signaaltjes op het juiste moment. Zie ook: https://www.youtube.com/watch?v=r7P2y__hD5M



Een mooi overzicht van wat de computer in een wasmachine doet zie je in het plaatje hieronder. De computer is de Control Unit. Die krijgt informatie van verschillende sensoren (waterniveau, positie van de deur, water temperatuur, snelheid van de trommel) en stuurt de output aan (de pomp, waterkleppen, het verwarmingselement en de motor).
Onder in het plaatje zie je drie feedbackpijlen staan. De feedback zorgt ervoor dat de kleppen, verwarming en motor aan- en uitgaan als de bijbehorende sensor een bepaalde waarde heeft bereikt. Een waterklep blijft bijvoorbeeld open staan totdat de waterniveausensor aangeeft dat er genoeg water in de wasmachine zit.

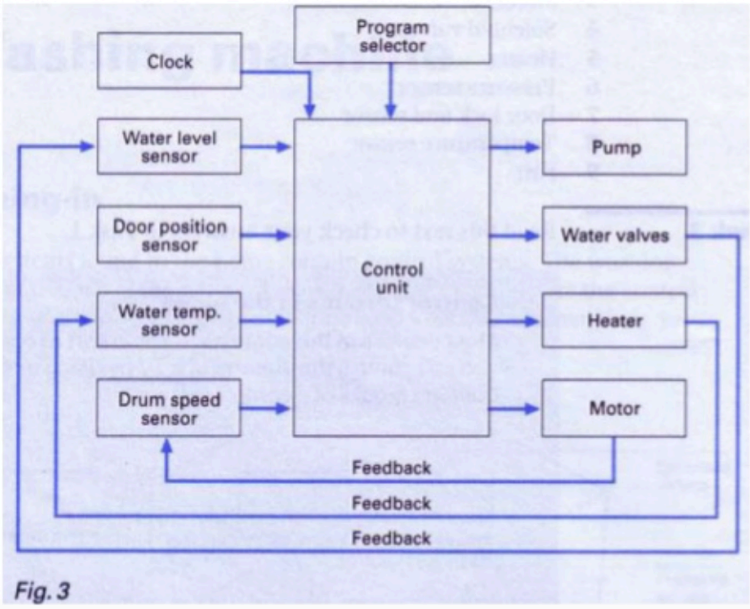


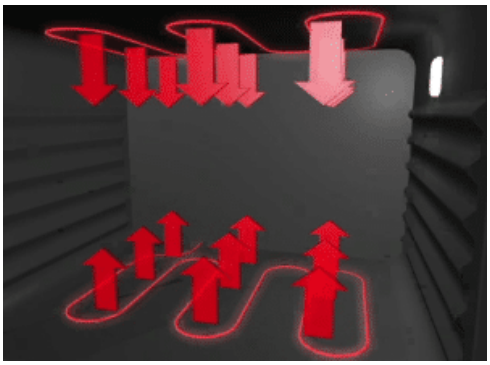
Fig. 3

Source: P. Fowler and M. Horsley, 'Contml systems in the home', CDT: Technology

Oven	In een oven wordt voedsel verhit, gebakken of gedroogd. Het voedsel kan op verschillende manieren verwarmd/ gegaard worden: vuur, lucht of straling. Een gasoven is een voorbeeld waarin het eten

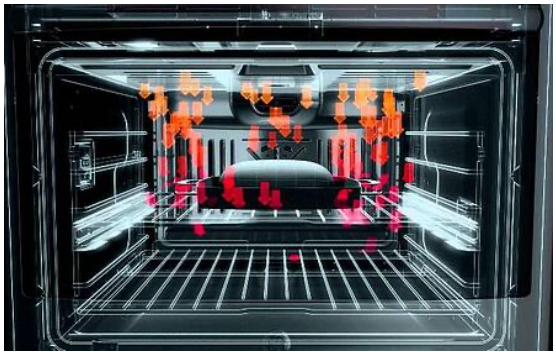
door middel van vlammen wordt verhit. In een heteluchtoven wordt de lucht verwarmd met verwarmingselementen en rondgeblazen. In een magnetron verhit elektromagnetische straling het voedsel.

Oven met verwarmingselementen die eten van meerdere kanten verwarmd.

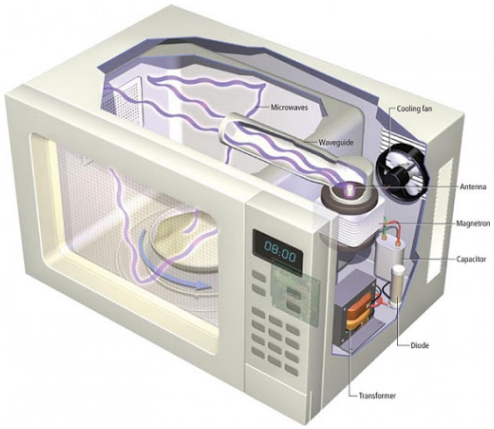
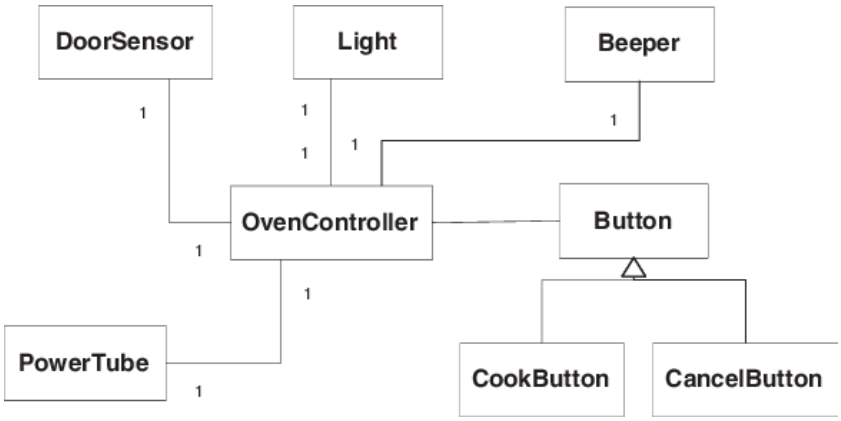


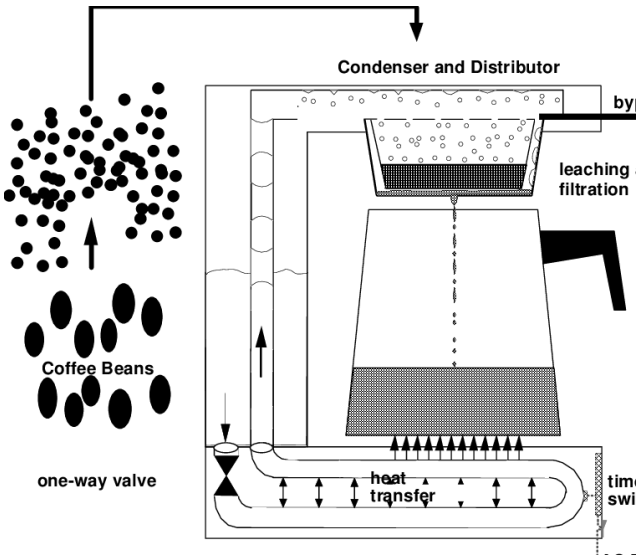
Oven waarin de warme lucht door een ventilator wordt rond geblazen.

Bij onderstaande oven zit een verwarmingselement aan de achterkant. Die verwarmd de lucht en die verwarmde lucht wordt door een ventilator de oven ingeblazen.



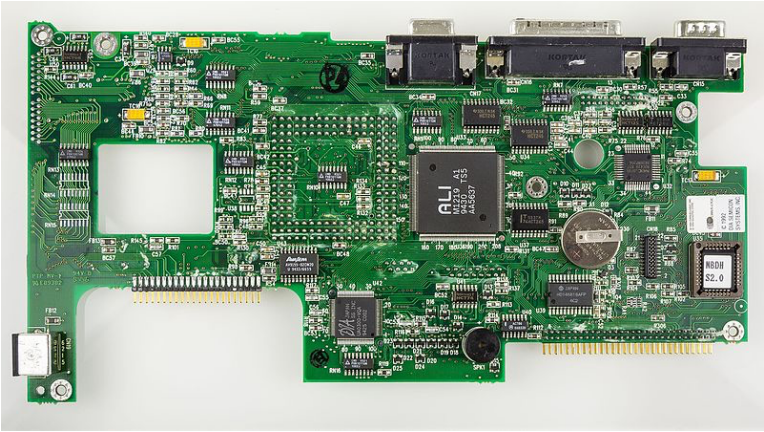
In een magnetronoven verhit elektromagnetische straling het eten. In het Engels noem je zo'n oven een microwave oven. Microwave of

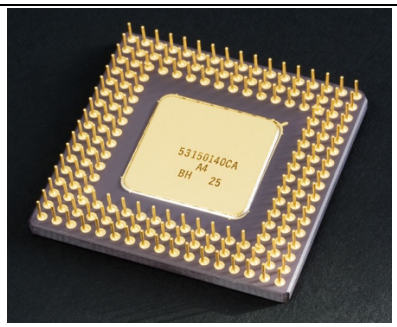
	in het Nederlands microgolven verhitten het eten. Het element dat de microgolven maakt is de daadwerkelijke magnetron. In Nederland hebben we dus het apparaat naar het element genoemd dat de microgolven maakt.  De computer in een oven of magnetron kan onder andere vaststellen of de deur dicht is, kan het licht aandoen, een pieper af laten gaan. Het kan ook bepalen welk programma afgewerkt moet worden. 
Koffiezetapparaat	Om koffie te maken met een koffiezetapparaat giet je water in de tank van het koffiezetapparaat. Het water stroomt naar een buis toe die leidt naar het gedeelte waar het water bij de koffie kan komen. Onder in het koffiezetapparaat zit een verwarmingselement. Een elektrische stroom verhit het verwarmingselement en die verhit het water. Door dat het water verwarmd wordt ontstaan er luchtbubbels en die nemen het water omhoog door de buis. Door een klep kan het water niet terug naar de tank maar alleen door de buis omhoog. Een temperatuursensor zorgt ervoor dat het water niet te heet wordt. Het hete water komt in de filter terecht waar het samen met de koffie in de koffiepot terechtkomt. Zie ook: https://www.youtube.com/watch?v=ce22H2-0xh4

<table>
<tr>
<td></td>
<td>


In een “gewone” koffiezetapparaat zit dus geen computer. Er zijn echter wel koffiezetapparaten die je kunt programmeren. De computer regelt dan bijvoorbeeld dat de koffie op een bepaalde tijd wordt gezet, wat voor soort koffie je wilt. Een voorbeeld van zo’n programmeerbare koffiezetapparaat is de Miele CM6150.


In dit filmpje wordt uitgelegd wat de mogelijkheden hiervan zijn: https://www.youtube.com/watch?v=7CKNFgIkSws
</td>
</tr>
<tr>
<td>Televisie</td>
<td>De meeste televisies zijn nu digitale televisies. Dat betekent dat de beelden en geluiden op een digitale manier worden verstuurd. Elektrische signalen werden eerst rechtstreeks doorgegeven. De elektrische signalen worden nu omgezet in een digitaal signaal van nullen en enen. Het voordeel hiervan is dat er geen informatie onderweg verloren gaat. Om de beelden en geluiden te ontvangen kun je gebruik maken een kabel of satelliet. Op de televisie zelf kun je gebruik maken van een digitaal menu waarin je het programma dat je wilt kijken kunt kiezen. Als je een programma wilt pauzeren, terugkijken of opnemen maak je ook gebruik van de computer (de</td>
</tr>
</table>

	beelden moeten worden opgeslagen). Via de televisie kun je ook gebruik maken van streamingdiensten zoals Netflix of Disney+. Het gebruik maken van streamingdiensten betekent dat je aangesloten moet zijn op een netwerk waarop de video's te bekijken zijn. Via de televisie kun je ook het internet op. Ook daarvoor heb je een computer nodig. Uitleg met filmpjes over werking van televisie: analoge televisie (zwart-wit): https://www.youtube.com/watch?v=-nC0U5rnO44 analoge televisie (kleur): https://www.youtube.com/watch?v=r38nVmxBfvM
Laptop	Een laptop is een van de meest voorkomende vormen waarin kinderen een computer tegen zullen komen. Een computer is een computer als je input kunt geven aan het apparaat (bijvoorbeeld via het toetsenbord, scherm, microfoon, videocamera of muis), die informatie opgeslagen en bewerkt (in de processor) kan worden en weergegeven kan worden. Belangrijke onderdelen van elke computer zijn het omhulsel waarin alles is opgenomen, de stroomvoorziening, het moederbord (een printplaat) waarop alle onderdelen zijn aangesloten, de processor die alle berekeningen uitvoert, de harde schijf waar alle informatie wordt opgeslagen, het RAM-geheugen dat de computer nodig heeft om informatie snel te gebruiken en een grafische kaart om alles mooi op je beeldscherm te krijgen. **Moederbord**  **Processor** (bovenkant en onderkant)



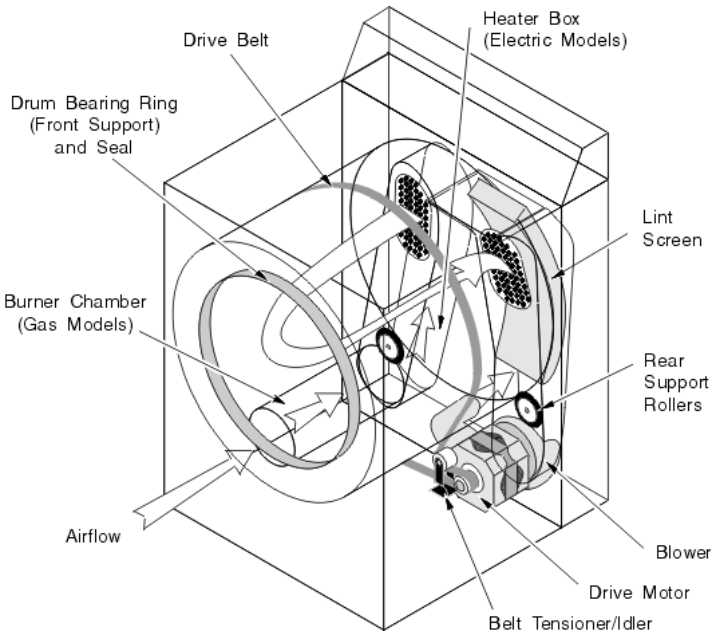
Harde schijf (twee voorbeelden)

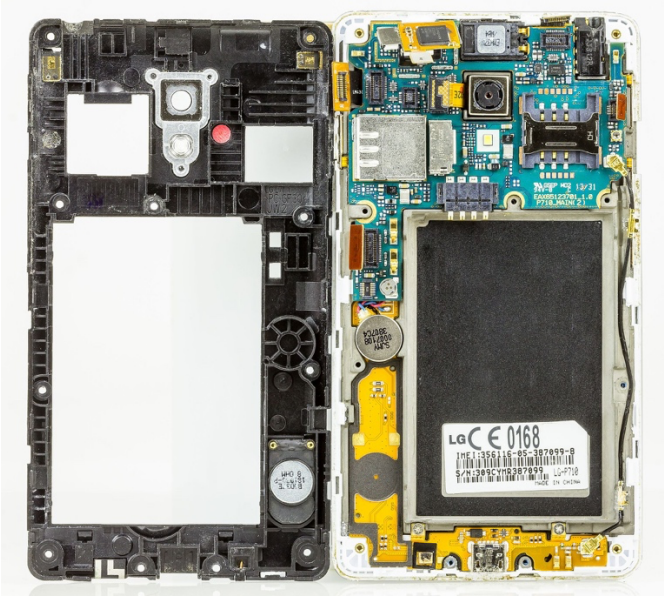

RAM-geheugen

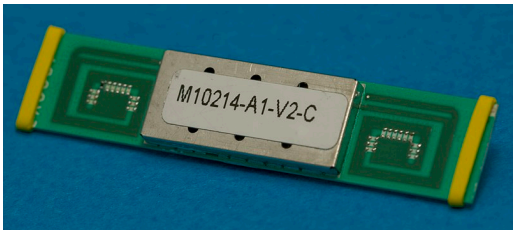
Grafische kaart

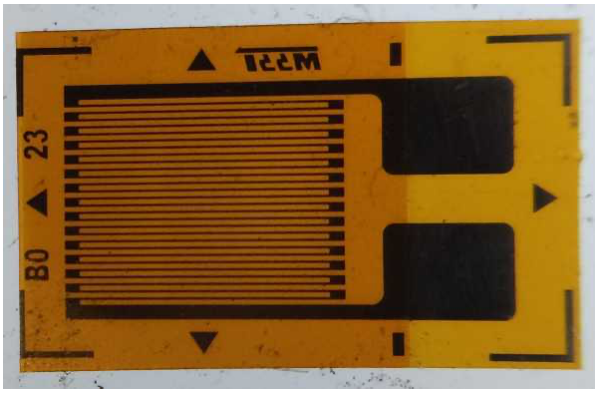
Een mooie videoreeks van code.org over wat een computer is en waar het uit bestaat: https://www.youtube.com/watch?v=mCq8-xTH7jA&list=PLzdnOPI1iJNcsRwJhvksEo1tJqjIqWbN-&index=2

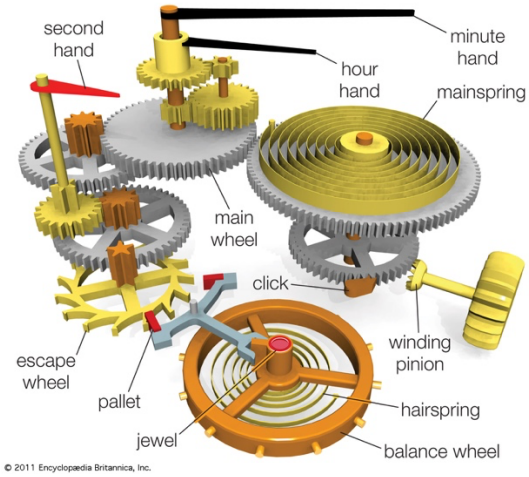
Video over wat er in een laptop zit (animatie): https://www.youtube.com/watch?v=HB4I2CgkcCo

	Video over hoe een computer werkt (animatie): https://www.youtube.com/watch?v=AkFi90lZmXA Video over wat er in een laptop zit (video): https://www.youtube.com/watch?v=ExxFxD4OSZ0 https://www.youtube.com/watch?v=O9vO_CVNXlg
Wasdroger	Een wasdroger maakt de was droog door de was rond te laten draaien in een trommel, lucht te verwarmen en de verwarmde lucht door de trommel te blazen. Een droger kan de verwarmde de lucht uit de droger blazen waarmee ook het vocht verdwijnt of het vocht opvangen in een grote bak. Een moderne droger gebruikt een computer om het juiste programma mee in te stellen. Het kan ook de sensoren (temperatuur en vochtigheid) gebruiken om vast te stellen hoe droog de was al is.  Video hoe een droger werkt: https://www.youtube.com/watch?v=UC5I5fmruCc
Mobiele telefoon	Een moderne mobiele telefoon is een combinatie van een telefoon, computer, fotocamera en gps-apparaat. Net zoals de laptop zullen de meeste leerlingen dit associëren met een computer. En net als de laptop zijn de onderdelen die dit een computer maken aanwezig: je input kunt geven aan het apparaat, je kunt informatie opslaan, bewerken en weergeven. De **telefoon** in de mobiele telefoon zet je stem om in een digitaal signaal (bestaande uit nullen en enen). Dit signaal wordt als een elektromagnetisch signaal verstuurd. Met behulp van zendmasten wordt het signaal, via kabels, doorgegeven naar een zendmast in de buurt van degene die je wilt bellen. Die zendmast stuurt dan weer

	een elektromagnetisch signaal naar de andere telefoons. In de telefoon wordt de nullen en enen in het elektromagnetische signaal weer omgezet naar je stem. De **fotocamera** in je mobiele telefoon maakt gebruikt van een sensor (heet een CCD). Elektromagnetische straling wordt omgezet in een elektrische lading. Die elektrische lading wordt weer gedigitaliseerd (omgezet in nullen en enen) en kan zo worden opgeslagen in het geheugen. Het **GPS-apparaat** in je mobiele telefoon maakt gebruikt van satellieten, grondstations en ontvangers. GPS staat voor Global Position System. Oftewel: wereldwijd plaatsbepalingssysteem. Je mobiele telefoon stuurt een radio-signaal naar een satelliet. Die kijkt hoe ver andere satellieten verwijderd zijn van jou. Door deze gegevens met elkaar te combineren kan de precieze plaats bepaald worden. Op verschillende plekken in dit systeem wordt de computer gebruikt. Bijvoorbeeld om de berekeningen van de afstand te maken en als de gps-positie wordt weergegeven op een kaart (zoals Google Maps). **Iphone van de binnenkant**  **CCD-chip van mobiele telefoon** 

	GPS (ontvangen en antenne) in mobiele telefoon  Video over hoe je mobiele telefoon ervan binnen uitziet: https://www.youtube.com/watch?v=fCS8jGc3log Video over uitleg bellen met je mobiele telefoon: https://www.youtube.com/watch?v=1JZG9x_VOwA Video over de CCD-sensor: https://www.youtube.com/watch?v=wsdmt0De8Hw Video over uitleg GPS: https://www.youtube.com/watch?v=wCcARVbL_Dk https://www.youtube.com/watch?v=FU_pY2sTwTA
Keukenweegschaal	Een keukenweegschaal kan meten hoe zwaar iets is en dat weergeven op een display. Dat kan analoog (dus zonder computer) en digitaal (dus met computer). Een analoge (of mechanische) keukenweegschaal gebruikt veren om de wijzer op het display uit te laten slaan.  

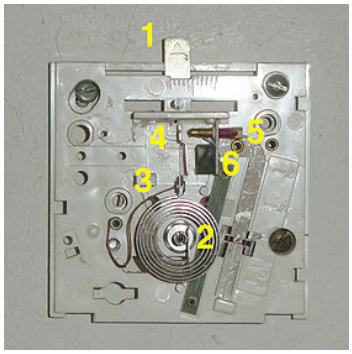
	  Een digitale weegschaal gebruikt een weegcel. Dat is een elektronische sensor. In de weegcel zit een rekstrookje. Dat rekstrookje rekt uit als je er een gewicht opzet. Door het uitrekken van het rekstrookje verandert het elektronische signaal dat de weegcel produceert. Die verandering wordt omgezet in een bepaald gewicht. Dit wordt digitaal weergegeven op een monitor. **Rekstrookje**  Video analoge weegschaal: https://www.youtube.com/watch?v=3SThvxCLk6c Video digitale weegschaal: https://www.youtube.com/watch?v=nnZj2IIULJ0
Wekker	Een wekker kan alleen mechanisch zijn maar ook digitaal. Een mechanische wekker werkt met tandwielen en springveren die met elkaar samenwerken om de tijd aan te geven.



Een digitale wekker werkt niet met tandwielen en veren maar gebruikt stroom om een zogenaamde kristaloscillator (of elektronische oscillator) aan te sturen. Een kristaloscillator is een stukje elektronica die een regelmatig signaal uitzendt als er stroom op staat. Dat regelmatige signaal is zo regelmatig dat je het kunt gebruiken in klokken en wekkers. Een chip leest de signalen die het krijgt en zet het om naar seconden, minuten en uren. Een apart programma zet het dan weer om naar cijfers op het scherm.

Buitenkant en binnenkant van kristaloscillator

Video mechanische wekker:
https://www.youtube.com/watch?v=9_QsCLYs2mY
Video mechanische klok:

	https://www.youtube.com/watch?v=9_QsCLYs2mY https://youtu.be/lDZHlcjQnOo?t=101 Video digitale klok/wekker: https://www.youtube.com/watch?v=QAVs1Iqk4kI
Thermostaat	Een thermostaat is een schakelaar die ervoor zorgt dat het in een ruimte een van tevoren ingestelde temperatuur blijft. Het schakelt het verwarmingssysteem van een huis in. Je hebt mechanische thermostaten en digitale thermostaten. Een mechanische thermostaat kan werken met een bimetaal. Bimetaal is een strip van twee verschillende soorten metaal. Als dit wordt verwarmd/ afgekoeld buigt het (in meer of mindere mate). Doordat het buigt maakt het contact en kan een stroompje worden doorgegeven. Als de stroom wordt doorgegeven dan gaat het verwarmingssysteem in huis aan. Is het warm genoeg dan buigt het bimetaal weer terug en wordt de stroomkring onderbroken. Binnenkant van een mechanische thermostaat die werkt met een bimetaal. De opgerolde strook metaal is het bimetaal.  In een digitale thermostaat wordt een elektronische weerstand gebruikt die reageert op de temperatuur. Deze elektronische weerstand noem je een thermistor. Afhankelijk van de temperatuur is de weerstand hoger of lager. En de computer is geprogrammeerd om die verandering in weerstand om te zetten naar een bepaalde temperatuur. Verder kan op basis van de tijd van de dag een specifieke actie worden ingesteld. Bijvoorbeeld dat als je opstaat de thermostaat op 20 graden Celsius moet komen te staan. **Thermistors**

	 Video uitleg werking bimetaal in een thermostaat: https://www.youtube.com/watch?v=6r9UAdb2kDo Video over thermistors: https://www.youtube.com/watch?v=D2SWOVWMdU8 https://www.youtube.com/watch?v=bjt4CrRL8yM