Universiteit Leiden
The Netherlands

Informatica & Economie

Programmeermisconcepties in Hedy

Ricardo Schaaf

Begeleid door:

Dr. ir. F.F.J. Hermans

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

23/03/2021

Abstract

De wereld wordt steeds digitaler en mede door deze verandering bieden steeds meer scholen programmeeronderwijs aan. Hierdoor maken kinderen op steeds jongere leeftijd kennis met programmeren. Het leren van een programmeertaal kan een grote uitdaging zijn, en om hierbij te helpen zijn er specifieke programmeeromgevingen gecreëerd om kinderen te leren programmeren. Het kan het voorkomen dat kinderen tijdens het aanleren van een programmeertaal een verkeerd begrip hebben gekregen van een programmeerconcept. In deze gevallen spreken we van een programmeermisconceptie. Een programmeermisconceptie kan ontstaan door de semantiek van natuurlijke taal, wiskundige kennis of het hebben van een verkeerd mentaal model. Voor deze scriptie hebben we een literatuuronderzoek uitgevoerd waarbij er gekeken wordt naar bestaande programmeermisconcepties die voorkomen in andere programmeertaal en in welke vorm deze kunnen voorkomen in Hedy. Daarnaast is er ook gekeken naar de huidige staat van programmeeronderwijs voor kinderen en welke vaardigheden er naast programmeren worden aangeleerd om te kunnen functioneren in de digitale samenleving. Deze thesis laat een overzicht zien van bestaande programmeermisconcepties die mogelijk voorkomen in Hedy, en eindigt met een onderzoeksopzet en specifieke opdrachten die bedoeld zijn om bestaande programmeermisconcepties in Hedy te vinden.

Inhoud

1. Introductie

De interesse in programmeeronderwijs is in de afgelopen decennia sterk gegroeid. Programmeren is in de afgelopen jaren in veel curricula van verschillende leeftijden geïntroduceerd. Het is een grote uitdaging voor leerlingen om een nieuwe programmeertaal te leren zonder voorkennis van programmeren [1]. Dit komt door de vele concepten, regels en beperkingen die bij programmeren komen kijken. Voor beginnende programmeurs kan deze hoeveelheid aan nieuwe informatie overweldigend zijn.

De traditionele onderwijsaanpak en de moeilijke programmeerconcepten verminderen de motivatie van studenten en hun interesse om het programmeren te leren [2]. Om deze problemen te overwinnen zijn er verschillende benaderingen en soorten educatieve programmeeromgevingen voorgesteld. Het gebruik maken van educatieve games in het leerproces is hier een voorbeeld van [3].

Het ontstaan van misconcepties over programmeerconcepten is een ander probleem wat speelt bij beginnende programmeurs. Een misconceptie zorgt ervoor dat een leerling een bepaald programmeerconcept verkeerd interpreteert of onjuist gebruikt. Door programmeermisconcepties van beginnen programmeurs te begrijpen kan men het onderwijs aanpassen en de studenten in de juiste richting sturen, wat de kwaliteit van het onderwijs positief kan beïnvloeden [4].

Er hebben veel onderzoeken plaatsgevonden naar misconcepties van onervaren programmeurs en kinderen voor specifieke statements of stukken code. In deze thesis focussen wij ons op de programmeermisconcepties van kinderen die voor komen in de programmeertaal Hedy.

1.1 Onderzoeksvraag

Het doel van deze thesis is om te kijken welke misconcepties er voorkomen bij kinderen in de programmeertaal Hedy. Door een overzicht te bieden van hoe misconcepties voorkomen bij kinderen hopen wij docenten te helpen met het begrijpen van de moeilijkheden die kinderen ondervinden bij het leren van een programmeertaal. De onderzoeksvraag van deze thesis luidt:

RQ: Hoe komen bekende programmeermisconcepties voor in de programmeertaal Hedy?

Om deze vraag te beantwoorden zullen we eerdere onderzoeken naar misconcepties in programmeertalen bestuderen en kijken naar wat misconcepties veroorzaakt. Vervolgens zullen we een kwalitatief onderzoek uitvoeren of wij misconcepties kunnen vinden die kinderen hebben tijdens het leren van de programmeertaal Hedy.

1.2 Opbouw

Deze thesis begint met een achtergrond onderzoek naar programmeeronderwijs, computational thinking en Hedy. Vervolgens zullen we kijken welke misconcepties er voorkomende in de literatuur en wat de oorzaken van deze misconcepties zijn. We zullen aan de hand van de literatuur en eigen onderzoek uiteenzetten welke misconcepties er voor kunnen komen in Hedy en op welke manier. De thesis eindigt met een opzet voor een onderzoek naar programmeermisconcepties in Hedy.

2. Achtergrond

2.1 De groeiende vraag naar programmeurs

Volgens de cijfers van het U.S. Bureau of Labor Statistics zal de werkgelegenheid van softwareontwikkelaars naar verwachting met 22 procent groeien tussen 2019 en 2029 [5]. Dit is veel hoger dan het gemiddelde voor alle beroepen. Hoe komt het dat de vraag naar softwareontwikkelaars zo hoog is?

Je hoeft maar om je heen te kijken en je ziet het werk van softwareontwikkelaars: elke app die wordt ontwikkeld, elke nieuwe digitale library die wordt toegevoegd en elke nieuwe digitale service is het werk van softwareontwikkelaars. Elke keer dat er iets nieuws wordt ontwikkeld wordt ook de complexiteit van het digitale ecosysteem vergroot. Voor elke innovatieve oplossing die online komt neemt het werk om die oplossing en alle afhankelijke systemen ervan te onderhouden toe.

Deze beweging lijkt niet te stoppen. Hedendaags heeft bijna elk bedrijf een eigen website of maakt gebruik van online adverteren. Er bestaat ook veel competitie tussen softwarebedrijven. Denk hierbij aan de vele berichten apps zoals WhatsApp, Facebook Messenger en Telegram die allemaal dezelfde service aanbieden. Deze onderlinge competitie zorgt ervoor dat veel software meer dan 1 keer wordt gebouwd. [6]

Wanneer de software eenmaal gebouwd is lijkt het alsof deze dan ook 'klaar' is. De meeste regels code worden echter voortdurend gewijzigd en de meeste bedrijven blijven innoveren. Elke regel code heeft een levensduur van slechts een paar jaar [7]. Maar gedurende deze levensduur moet de software onderhouden worden, dit gebeurt ook door programmeurs. Deze onderhoudskosten zijn vaak de grootste uitgaven voor een nieuwe app/website [8].

Bovenstaande redenen hebben er in de afgelopen jaren voor gezorgd dat de vraag naar softwareontwikkelaars is gegroeid en deze trend lijkt zich voort te zetten. Om aan de groeiende vraag te voldoen zal het aanbod van softwareontwikkelaars me moeten groeien.

2.2 Focus op kinderen

Vanaf het moment dat computers publiekelijke beschikbaar waren werd er geprobeerd om kinderen te leren programmeren. Papert probeerde met zijn boek '*Mindstorms'* en zijn programmeertaal Logo de structuur van het onderwijs en de manier van leren te veranderen. [9] Er kwam geen grote verandering in het onderwijs en de meeste scholen gingen computers op een andere manier gebruiken. Er zijn veel programmeertalen en omgeving gemaakt met de intentie om programmeren toegankelijker te maken voor een groot aantal mensen, maar tot aan het begin van dit decennium bereikte dit niet zijn doel. [10]

Door de groeiende vraag naar softwareontwikkelaars, en om in te spelen op de groeiende invloed van techniek in de huidige maatschappij zijn veel landen Informatica (computer science) gaan opnemen in het school curriculum. Zo is programmeren opgenomen in het school curriculum van basisscholen in Australië [11] en het Verenigd Koninkrijk [12]. Denemarken [13] en Nieuw-Zeeland [14] hebben het opgenomen in het curriculum van de middelbare scholen.

De Europese Unie is hier op doorgaan is een onderzoek gestart om een beeld te geven van de huidige status en ontwikkeling rondom het introduceren en vestigen van informatica als een schoolvak in heel Europa [15]. Uit dit onderzoek is in 2017 het initiatief "Informatics for All" gelanceerd.

Het doel van "Informatics for All" is om Informatica een essentiële discipline te maken voor iedereen, een onderwerp dat beschikbaar is op alle niveaus in het onderwijssysteem. Door het leren van

informatica zullen de studenten in staat zijn om de ontwikkeling van de digitale wereld beter te begrijpen en om eraan deel te nemen. Het zal ook bijdragen aan het opleiden en werven van het grote aantal IT-specialisten die Europa nodig heeft om zijn positie in de digitale wereldeconomie te behouden en te verbeteren. [16]

In 2016 lanceerde president Obama een soortgelijk initiatief "CS For All" met als doel: *" to empower all American students from kindergarten through high school to learn computer science and be equipped with the computational thinking skills they need to be creators in the digital economy, not just consumers, and to be active citizens in our technology-driven world."* [17].

Bovenstaande voorbeelden geven aan dat de focus van programmeeronderwijs steeds meer komt te liggen op kinderen, maar is programmeren niet te lastig voor kinderen?

2.3 Is programmeren moeilijk?

"*Learning to program is notoriously difficult. Substantial failure rates plague introductory programming courses the world over, and have increased rather than decreased over the years"* [18]. Dit is een van de vele onderzoeken die stelt dat leren programmeren moeilijk is. Bosse & Gerosa, Jenkins en Robins et al zeggen ondersteunen deze mening [19]–[21]. Bornat & Dehnadi stellen in dit onderzoek ook dat programmeersonderwijs voor beginners last heeft van hoge uitvalpercentages dat in de afgelopen jaren is gestegen. De hoge uitvalpercentages zijn vaak genoemde motivaties om een onderzoek naar programmeeronderwijs te starten [22], [23]. Waar komt de overtuiging dat programmeren moeilijk is en er hoge uitvalpercentages zijn vandaan?

Deze lijken niet te baseren op een statistiek of een degelijk onderzoek naar de onderwerpen. In 2007 hebben Bennedsen en Caspersen een onderzoek uitgevoerd met als doel om het gemiddelde faal- en slagingspercentage voor Inleidende programmeerlessen op universiteiten te vinden. Het onderzoek bestond uit het versturen van een enquête naar verschillende onderwijsinstelling met de vraag wat de slagings-, mislukkings-, overslaan-, en afbreekpercentages waren. Hierbij werd er ook nog onderscheid gemaakt tussen hogescholen en universiteiten en welke type lessen het waren. De uitkomst was dat 67% van de studenten het vak halen. Bennedsen en Caspersen trokken hieruit de conclusie dat het uitvalpercentage van programmeeronderwijs niet alarmerend hoog was [1]. Watson en Li hebben zeven jaar later eenzelfde studie uitgevoerd met een data-analyse van 161 inleidende programmeercursussen uit 15 verschillende landen. Ze bevestigde de bevindingen van Bennedsen en Caspersen omdat er uit hun onderzoek een gemiddelde slagingspercentage van 67,6% kwam [24].

Nu, meer dan tien jaar later, ontvangt de studie van Bennedsen en Caspersen nog steeds veel [25] citaten. Het hoge aantal van citaten in de afgelopen jaren geeft aan dat het uitvalpercentage een relevant onderwerp is voor veel onderzoekers. De meeste artikelen/papers gebruiken het onderzoek om te beweren dat inleidende programmeervakken een bepaald probleem hebben en suggereren vervolgens een interventie om het waargenomen probleem te verlichten [26]. Omdat het onderwerp nog steeds relevant is en programmeeronderwijs steeds wijd geaccepteerder wordt zijn Bennedsen en Caspersen teruggekeerd om het onderzoek nogmaals uit te voeren. In dit onderzoek kwamen ze tot dezelfde conclusie: het uitvalpercentage van programmeeronderwijs is niet alarmerend hoog. Ten opzichte van hun eerder uitgevoerde onderzoek is het zelfs gedaald met vijf procent [26]. Maar waarom is het van belang of de aanname dat programmeren moeilijk is klopt of niet?

De aanname dat programmeren moeilijk is kan gevolgen hebben voor de kwaliteit van programmeervakken. Een gevaar van deze aanname is dat een docent kan denken dat het hoge uitvalpercentage komt omdat dit 'normaal' is voor programmeervakken en zal zodoende geen actie ondernemen om de manier van lesgeven te verbeteren [27].

Zo vinden veel docenten het lastig om studenten logisch redeneren aan te leren. Voor de meeste docenten gaat het structureren en oplossen van een probleem automatisch in hun hoofd en vinden ze het moeilijk om exact uit te leggen wat er gebeurt. Ze hebben geprobeerd om logisch redeneren uit te leggen met pseudocode, maar de meeste docenten hebben het aanleren opgegeven [19]. Maar de aanname dat programmeren moeilijk is heeft er ook voor gezorgd dat er veel verschillende leermethodes zijn ontwikkeld [28] om het aanleren van programmeren te verbeteren. De onderzoeken [29] naar de meest effectieve manier om programmeren aan te leren en het blijven ontwikkelen van nieuwe leermethodes [30] laten zien dat docenten en instanties zich bewust zijn van de gevaren die gepaard kunnen gaan met de aanname dat programmeren moeilijk is.

De aanname dat programmeren moeilijk is kan ook gevolgen hebben voor studenten. Zo kan het potentiële studenten afschrikken om informatica te gaan studeren of nadelig werken voor de zelfeffectiviteit van huidige en toekomstige studenten. De zelfeffectiviteit is "*het oordeel van een individu over zijn of haar vermogen om een taak binnen een specifiek domein uit te voeren"* [31]. Als programmeren als moeilijk wordt neergezet kan dit ervoor zorgen dat de zelfeffectiviteit van studenten verminderd en wat zorgt voor mindere prestaties.

Bovenstaande alinea's laat zien dat de aanname dat programmeren moeilijk is gevolgen kan hebben voor de kwaliteit van programmeren, maar dat dit niet het geval hoeft te zijn als docenten en studenten zich hiervan bewust zijn.

2.4 Huidige situatie van programmeeronderwijs.

Kennisnet, een gerespecteerde dienstverlener op het gebied van ICT en onderwijs, kreeg meer en meer de vraag van scholen in het funderend onderwijs wat ze moesten doen met programmeren in de klas. De in Nederland gestarte initiatieven zoals 'Programmeren en coderen in de klas' [32] en het 'Programmeerplein' van NOT 2019 [33] waren niet meer genoeg om aan de vraag van scholen te voldoen. Scholen willen programmeren in hun onderwijs integreren. Andere scholen vinden enkel leren programmeren te beperkt en willen hun leerlingen digitale geletterdheid aanleren. Maar hoe doe je dat? In Engeland is programmeren toevoegt aan het verplichte curriculum. Door deze invoering onder de loep te nemen kunnen we kijken welke stappen Nederland al heeft gemaakt en nog moet maken voor een verbeterde digitale generatie.

2.4.1 Programmeeronderwijs in Engeland

De Engelse overheid heeft als doel gesteld dat leerlingen van de basisscholen het vermogen moeten leren ontwikkelen om hun kennis van computers en hun eigen creativiteit te gebruiken om de wereld te begrijpen en eventueel digitaal naar eigen hand te zetten [34].

Vanaf 2014 is het computing-onderwijs officieel deel van het Engelse National Curriculum. Het doel van computing-onderwijs is dat leerlingen het vermogen ontwikkelen om computerkennis en creativiteit te gebruiken om de wereld te begrijpen, en waar mogelijk (digitaal) naar hun hand te zetten. Er wordt gemiddeld een uur per week aandacht besteed aan computing-onderwijs [34]. Een van de manieren om dit te bereiken ligt in een van de grondbeginselen van informatica namelijk computational thinking. Dat betekent: begrijpen hoe digitale systemen werken en hoe je ze kunt inzetten door middel van programmeren. Het computing-onderwijs richt zich meer hoe computers en computersystemen werken en heeft het bestaande ICT-onderwijs vervangen. De handleiding

‘Computing in the national curriculum: A guide for primary teachers’[35] wordt veel gebruikt door scholen in Engeland.

Een jaar later blikken Stiller en Pijpers terug op het verloop van de invoering van programmeeronderwijs op de basisschool in Engeland [36]. Hieruit blijkt dat het merendeel van de leerlingen enthousiast is over het nieuwe curriculum. Dit kwam omdat het erg aansluit bij de vaardigheden van het ‘moderne technologische kind’. Leerlingen kunnen met de nieuwe stof beter laten zien wat ze gemaakt hebben. De leerkrachten zijn minder enthousiast over het computing-onderwijs. Terwijl de leerkrachten juist een kernfactor zijn om dit nieuwe curriculum te doen slagen. Veel leerkrachten hebben te weinig zelfvertrouwen in hun eigen vaardigheden om het vak computing goed te kunnen geven [37]. Het gebrek aan kennis en vaardigheden omtrent het vak computing wordt veroorzaakt door een gebrek aan training en ondersteuning [38]. Het belang van een toegewijde lerarenopleiding is al eerder aangetoond [39]. Een advies dat hiervoor wordt genoemd is het in kaart brengen van de huidige kennis en vaardigheden van de leerkracht en daar een plan van aanpak op baseren.

2.4.2 Programmeeronderwijs in Nederland

Uit Engeland blijkt dat dat het programmeeronderwijs goed is ontvangen door kinderen. Voor het invoeren van het programmeeronderwijs in Nederland zijn al verschillende stappen gezet door overheidsinstanties.

In 2013 publiceerde de Koninklijke Nederlandse Akademie van Wetenschappen (KNAW) een advies voor Informatica in het voortgezet onderwijs. Dit advies was beïnvloed door een invloedrijk rapport van The Royal Society [40] in het Verenigd Koninkrijk. De op dat momenten gegeven vakken: Informatiekunde en Informatica bereidden de kinderen niet goed voor op de huidige digitale maatschappij. Dit advies/rapport heeft voor een algehele herziening gezorgd van het Nederlandse voortgezet onderwijs in digitale informatie en communicatie. Ze benadrukken dat in de ontwikkeling van de maatschappij de omgang met informatie centraal staat. De term i*nformatiemaatschappij* wordt gebruikt voor de 21ste eeuw. Mensen en apparaten wisselen gegevens door middel van tekst, audio en video uit in gigantische hoeveelheden en via multimediale kanalen. De explosie in informatie-uitwisseling heeft ervoor gezorgd dat de wetenschap *data driven* is geworden. Informatie over files, weg- en weercondities sturen de vervoersector. Informatie over bedrijfs- en consumentenvertrouwen, beurskoersen en kwartaalcijfers sturen de financiële sector. Informatie over onlinesurfgedrag en aankopen sturen de reclame en marketingsector. Iedereen maakt tegenwoordig deel uit van de *informatiemaatschappij* als consument van informatie, als producent of als schakel in het informatienetwerk.

De KNAW constateert dat het welvaren van een individu en van de maatschappij die door deze individuen wordt gevormd, direct afhangt van de wijze waarop allen effectief, efficiënt en weloverwogen met de informatie omgaan. Ze vinden de vaardigheden in de omgang met digitale informatie even onmisbaar als vaardigheden in taal en rekenen [41], ook noemen ze de nieuwe vaardigheden digitale geletterdheid. Het belang van digitale geletterdheid wordt ondersteunt door eerder onderzoek [42].

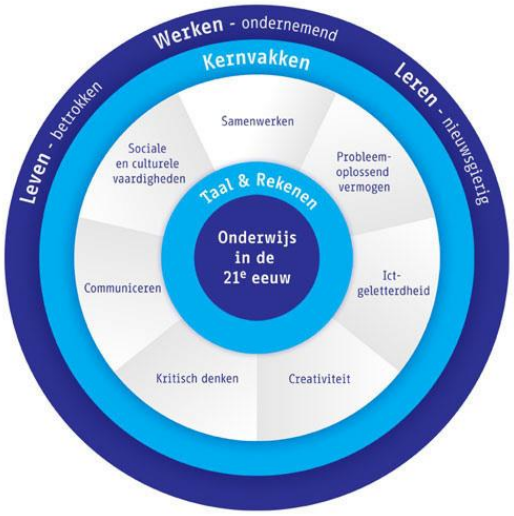
Het rapport sloot af met een advies om een verplicht van Informatie en communicatie te ontwikkelen en in te voeren voor het curriculum van de onderbouw van het voortgezet onderwijs en het keuzevak informatica in het hoger secundair onderwijs grondig te vernieuwen. Hierin werd ook specifiek het belang van computational thinking genoemd [43].

Als reactie op dit rapport heeft het ministerie van Onderwijs, Cultuur en Wetenschap het SLO gevraagd om te onderzoeken wat deze vaardigheden precies inhouden en in hoeverre ze aandacht moeten krijgen in het funderend onderwijs. Daarbij is specifiek gevraagd om aandacht te besteden aan digitale geletterdheid [44].

2.5 21ste eeuw vaardigheden

Het SLO heeft de vaardigheden die nodig zijn om goed gebruik te maken van met digitale informatie beter gedefinieerd. De vaardigheden gaan over wat leerlingen van nu, in de 21ste eeuw, nodig hebben om te overleven en te excelleren. Een veelgebruikte verzamelnaam voor de vaardigheden is '21st Century Skills' oftewel 21e-eeuwse vaardigheden. Er zijn verschillende modellen te vinden in de literatuur met ieder zijn eigen invulling van de vaardigheden die nodig zijn voor de toekomst. Voogt en Roblin [42] hebben deze modellen onderzocht en kwamen tot de conclusie dat de volgende vaardigheden in alle modellen worden genoemd: Samenwerking, communicatie, ICT geletterdheid, sociale en/of culturele vaardigheden. Er wordt aan toegevoegd dat creativiteit, kritisch denken en probleem oplosvaardigheden in bijna alle modellen wordt genoemd. Het SLO heeft dit onderzoek gebruikt om tot de volgende definitie van het begrip 21 -eeuwse vaardigheden te komen:

"*Generieke vaardigheden en daaraan te koppelen kennis, inzicht en houdingen die nodig zijn om te kunnen functioneren in en bij te dragen aan de 21e -eeuwse samenleven.*" [44]



Figuur 1 Oud model 21e eeuwse vaardigheden

Figuur 2 Het model voor 21e eeuwse vaardigheden ontwikkeld door SLO en Kennisnet

Bij de definitie werd in het initiële rapport Figuur 1 toegevoegd. Na 2 jaar is het model aangepast naar Figuur 2. In het nieuwe model is ICT-geletterdheid opgesplitst in 4 vaardigheden namelijk: mediawijsheid, ICT-basisvaardigheden, informatievaardigheden en Computational thinking. De combinatie van deze vier vaardigheden wordt gebruikt als definitie voor digitale geletterdheid.

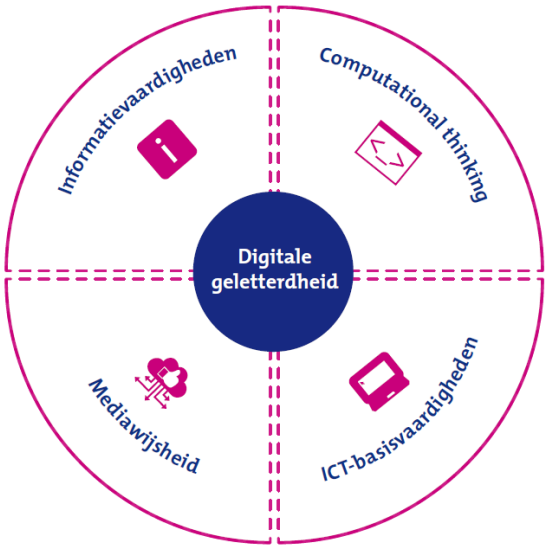
2.6 Digitale Geletterdheid

Digitale geletterdheid is een van de vaardigheden die in alle onderzoeken terugkomt [41], [42], [45], vaak wordt de term digitale geletterdheid genoemd, maar de termen ICT-vaardigheid, *information literacy* worden hier ook voor gebruikt. Het KNAW waarschuwde in 2013 al voor het ontstaan van een kloof tussen digitaal geletterden en digitaal ongeletterden [41]. Kinderen moeten niet alleen 'gewoon' leren lezen en schrijven. Ze moeten ook leren effectief, veilig, kritisch en bewust gebruik maken van de digitale mogelijkheden die momenteel beschikbaar zijn. Ze moeten zich ook voorbereiden op de digitale toekomst: op school, bij een vervolgopleiding, op het werk, als consument en producent, als burger ten opzichte van de overheid en als burgers onder elkaar. Digitale geletterdheid heeft dit als doel: "Om leerlingen op eigen kracht te leren functioneren in een samenleving waarin digitale technologie en media een belangrijk plaats hebben" [46].

Sinds het rapport van SLO in 2014 heeft digitale geletterdheid een belangrijkere plek gekregen binnen scholen. De komende jaren zal het SLO-kerndoelen voor digitale geletterdheid ontwikkelen in samenwerking met leraren, vakdidactici, scholen en onderwijsorganisaties. Naar verwachting zijn die kerndoelen in 2024 helemaal klaar. Eind 2022 zullen de conceptkerndoelen, waarmee de scholen aan de slag kunnen, klaar zijn. Er zijn verschillende handboeken verschenen om de stap naar structurele implementatie van digitale geletterdheid te ondersteunen [45], [47].

Volgens deze handboeken, het SLO [44] en Curriculum.nu [48] is digitale geletterdheid een combinatie van 4 vaardigheden:

- ICT-basisvaardigheden: *De werking van computers en netwerken begrijpen, kunnen omgaan met verschillende soorten technologieën en de bediening ervan en de mogelijkheden en beperkingen van technologie begrijpen.*
- Computational thinking: *Problemen op een zodanige manier formuleren dat het mogelijk wordt een computer of ander digitaal gereedschap te gebruiken om het probleem op te lossen.*
- Informatievaardigheden: *Een informatiebehoefte kunnen signaleren en analyseren en op basis hiervan relevante informatie zoeken, selecteren, verwerken en gebruiken.*
- Mediawijsheid: *Kennis, vaardigheden en mentaliteit die nodig zijn om bewust, kritisch en actief om te gaan met media.*



Figuur 3 Digitale geletterdheid volgens kennisnet

Volgens Platform 2032 [49], een eindadvies van het Platform Onderwijs2032 over de kennis en vaardigheden die leerlingen opdoen met het oog op toekomstige ontwikkelingen op de samenleving, is Computational thinking de voornaamste vaardigheid die getraind kan worden met de leerlijn 'Programmeren in het PO'. Programmeren in het PO is een leerlijn die in 2016 gelanceerd is door de PO-raad in samenwerking met haar ICT-partner Kennisnet om begrippen en principes uit te leggen die horen bij programmeren [50].

Toch zitten, net als in Engeland, niet alle Nederlandse basisschoolleerkrachten te springen om meer te doen met computational thinking en programmeren. In het artikel "Digidenkers"[37] benadrukken Strijker (SLO), Meijer (Fier in Friesland) en 't Hart (O2G2) dat de ontwikkelde *unplugged*-leerlijn juist heel laagdrempelig is en goed aansluit bij al bestaande leeractiviteiten. Alle leraren kunnen met de *unplugged*-lijn uit de voeten en niet enkel degenen die computervaardig zijn. Dit komt omdat de *unplugged*-activiteiten activiteiten zijn waarbij leerlingen leren programmeren zonder dat ze daarbij voorkennis hoeven te hebben over onderwerpen als hardware, software of infrastructuur [50].

2.7 Computational thinking

Toen Seymour Papert in 1980 nadacht over het huidige onderwijssysteem en de manier waarop kinderen leerden gebruikte hij voor het eerst de term '*Computational thinking*' en suggereerde dat computers het denken en de manier van kennis verkrijgen kon verbeteren [9]. De Amerikaanse hoogleraar Wing herhaalde de ideeën van Papert en legde het concept Computational thinking verder uit. Ze liet ook de toepassingen van Computational thinking bij het oplossen van problemen zien [51]. Wing omschreef het concept als: "*Computational thinking is taking an approach to solving problems, designing systems and understanding human behaviour that draws on concepts fundamental to computing*".

Computational thinking wordt in het bijzonder gebruikt als een onderzoekende en probleemoplossende methode die computerwetenschappelijke concepten, hulpmiddelen en technieken in wetenschap, technologie, techniek en wiskunde gebruikt (STEM). Nieuwe geïntegreerde velden zoals computationele biologie, computationele chemie en computationele fysica zijn ontstaan die profiteren van de kracht van computation bij het uitbreiden van ontdekking en innovatie binnen de klassieke STEM-velden [52].

Computational thinking beïnvloedt onderzoek in binnen de huidige disciplines, zowel in de bètawetenschappen als in de geesteswetenschappen [53]. Bewijs van de invloeden in andere gebieden: In de statistiek heeft machine learning de Bayesian methode en het gebruik van probabilistische grafische modellen geautomatiseerd [54]. In de biologie heeft Computational thinking ervoor gezorgd dat veel computational modellen worden gebruikt om dynamische processen uit de naturen te presenteren [55]. Computational thinking is ook economie aan het veranderen, door een nieuwe veld van computational micro-economie te creëren zoals het vinden van de optimale donor voor een *n*-aantal niertransplantaties [56].

De grote invloed van Computational thinking op andere disciplines heeft gezorgd voor een toenemende interesse om basis- en middelbare scholieren te betrekken bij Computational thinking. Het wordt genoemd als een essentiële competentie die moet worden opgenomen in de vaardigheden van elke leerling [57]. Het integreren van computational thinking in het onderwijs is nog relatief nieuw en er wordt veel onderzoek verricht naar projecten om leerlingen zo vroeg mogelijk kennis te laten maken met computational thinking [58].

Wanneer computational thinking langzaam in alle gebieden gebruikt gaat worden dan komt langzaam iedereen er ook mee in aanraking, dan wel direct dan wel indirect. Dit zorgt voor dezelfde vraag als die digitale geletterdheid/ 21e eeuwse vaardigenheden opdoet. Hoe zorgen we ervoor dat mensen deze vaardigheden leren en dat ze dit ook op de juiste manier leren zodat er geen misconcepties ontstaan.

2.8 Hedy

Omdat misconcepties het leerproces verstoren en wijdverspreid zijn onder studenten [59], wordt er veel onderzoek gedaan naar educatieve methodes en strategieën om misconcepties en andere problemen bij leerlingen weg te werken en zo de prestaties van leerlingen bij het leren programmeren te verbeteren. Een paar bevindingen op een rij: het gebruik maken van goede programmeervoorbeelden [60], het expliciet leren van programmeerstrategieën en -patronen [61], het ontwikkelen van een conceptinventarisatie [62] en peer instructie [63].

Naast deze methodes en strategieën gebruiken veel docententools of geavanceerde computertechnieken om specifieke problemen bij leerlingen aan te pakken. Zo zijn visualisatietools die de programmeerconcepten en de uitvoering van het programma illustreren nuttig geweest voor leerlingen om hun begrip te ontwikkelen van hoe computerprogramma's werken [64], [65]. Dit helpt bij het verbeteren van de onnauwkeurige mentale modellen van programmeerconcepten.

Een ander aspect waar veel tools zich op focussen is het leren van de syntax van programmeertalen. Het is belangrijk om hier op te focussen want wanneer leerlingen beginnen met het leren van programmeren dan zijn ze niet bekend met de nieuwe termen en de syntax van de programmeertaal. Hierdoor worden er fouten gemaakt in de syntax zoals het vergeten van haakjes, accolades en puntkomma's [66]–[68]. Eerder onderzoek heeft aangetoond dat leerlingen in 73% van de gevallen code met syntaxfouten inleverde, zelfs de beste studenten hadden in 50% van de gevallen syntaxfouten [69]. Er zijn heel veel nieuwe programmeeromgevingen ontwikkeld om helpen syntaxfouten te vinden of om deze zelfs te voorkomen [10].

De vorige onderzoeken kunnen opgedeeld worden in drie verschillende benaderingen om het leren programmeren voor beginners makkelijker te maken [70]: 1) mini of speel programmeertalen die specifiek op lesgeven zijn gericht zoals Papert's LOGO [9] of Scratch [71] 2) sub-talen van volledige programmeertalen zoals ProfessorJ [72] en 3) incrementele talen die beginnen met een subset van een taal en stapsgewijs nieuwe concepten toevoegen zoals DrScheme [73].

3. Misconcepties

Volgens psycholoog Piaget en de naar hem vernoemde Piaget-theorie zoeken kinderen naar begrip door interactie met de omgeving. Ze leren door de combinatie van eerdere ervaringen en de huidige mentale schema's die zijn ontwikkeld door assimilatie en aanpassing [74]. Ervaringen zorgen echter niet altijd voor de juiste interpretaties [75] wat als gevolg kan hebben dat een kind een misconceptie krijgt over een concept. Het probleem hiervan is dat als de misconceptie eenmaal gevestigd is in het hoofd van een kind het heel moeilijk is om deze te veranderen.

Longfield schrijft dit toe aan het feit dat mensen de neiging hebben om informatie vast te houden die overeenkomt met hun bestaande theorieën en inzichten, terwijl ze informatie negeren die deze theorieën tegenspreekt [76]. Ben-Ari stelt dat de afstand tussen het persoonlijke begrip van leerlingen van een concept en de wetenschappelijke visie van dit concept de bestaande misconcepties van leerlingen bepaalt [77]. Misconcepties worden dus ontwikkeld door de ervaringen van een persoon en vertegenwoordigen een onjuist begrip van een idee of concept. Dit is in de literatuur gekenmerkt als een misvatting, vooroordeel, naïef geloof, alternatieve conceptie of misverstand [59].

3.1 Programmeermisconceptie

Een programmeermisconceptie is het hebben van een onjuist begrip van een programmeerconcept of een reeks gerelateerde concepten. Programmeermisconcepties van studenten is een onderwerp dat al enige tijd wordt bestudeerd. Er worden verschillende factoren gegeven die mogelijk bijdragen aan misconcepties bij studenten. Een aantal factoren zijn: Natuurlijke taal, verkeerd mentaal model en wiskundige kennis, deze leggen we nu uit:

3.1.1 Natuurlijke taal

Commando's in een programmeertaal zijn meestal gebaseerd op natuurlijke taal. Hierdoor kan het de kennis van de natuurlijke taal een reden zijn van fouten en misconcepties van studenten [78]. In 1983 hebben Bayman & Mayer de misconcepties met betrekking tot individuele programeer statements in BASIC onderzocht. Een veelvoorkomende misconceptie bij de gebruikers was dat bij "INPUT A" daadwerkelijk de letter "A" opgeslagen werd, terwijl er daadwerkelijk een nummer in variabele "A" wordt opgeslagen [79].

Een andere oorzaak voor misconcepties is het construeren van de oplossing in een natuurlijke taal in plaats van in de programmeertaal. In het onderzoek van Spohrer en Soloway genereerde beginners voorwaardelijke statements eerst in het Engels in hun hoofd en voeren deze vervolgens in de computer in. Bijvoorbeeld: "The retry should be done when Problem_Type is not a, b, or c" De 'or' is in het Engels een correct gebruik maar in bepaalde programmeertalen, zoals Pascal, zorgen de Logica regels ervoor dat de computer een andere conclusie uit deze statement trekt [80].

Miller noemt een andere veelvoorkomende reden voor misconcepties: het gebruiken van een metonymie waarbij er verschil wordt gemaakt tussen de beoogde referent en de vermelde referent. Als iemand bijvoorbeeld vraagt: "pak het brood" dan bedoelt men "pak de zak met brood". Hierbij is het brood de vermelde referent, maar de beoogde referent is de zak met brood. Een mens kan het beoogde referent afleiden door zijn algemene kennis een computer kan dit niet. Als studenten gaan programmeren zullen ze echter wel expliciet naar de beoogde referent moeten verwijzen, anders zal er een fout optreden [78]. Ze moeten dus niet de metonymie gaan gebruiken zoals de dat doen in natuurlijke taal en communicatie en geloven dat de computer de beoogde referent kan afleiden.

3.1.2 Verkeerd mentaal model

Onvolledige of onnauwkeurige mentale modellen van computersystemen of code uitvoeringen kunnen ook leiden tot misconcepties [79], [81]. Een mentaal model wordt omschreven als de conceptie van de gebruiker over de 'onzichtbare' informatieverwerking die plaatsvindt in de computer tussen het moment van invoer en uitvoer [79].

Een van de dominantere hedendaagse leerstromingen is het constructivisme. Het constructivisme beweert dat kennis niet onafhankelijk van leerlingen kan bestaan. Leerlingen doen kennis op door de actieve wereld te combineren met bestaande cognitieve structuren in plaats van passief kennis te absorberen uit lezingen of studieboeken [77]. Het constructivisme benadrukt het belang van correcte mentale modellen in het programmeeronderwijs. Het leren van programmeren is meer dan het onthouden van specifieke feiten, regels en vaardigheden. Tijdens het leren construeert de student een mentaal model van de basis programmeerconcepten [81].

Syntaxfouten, zoals het missen van een haakje of een puntkomma, zijn duidelijk en makkelijk te herstellen. Een misconceptie als resultaat van een incompleet of verkeerd mentaal model zijn moeilijker te ontdekken en ook moeilijk om te veranderen. Dit komt omdat een student zijn mentale model vaak in het begin van het leren programmeren wordt gecreëerd [82]. Een voorbeeld hiervan is dat beginnende programmeurs aannemen dat de computer intelligent is. Pea noemt deze aanname een zogeheten 'superbug' waarbij de computer kan interpreteren wat de student wil bereiken en dit zelfstandig uitvoert [83]. Sleeman et al. ondersteunde deze bevinding. Sommige fouten kunnen worden verklaard doordat de gebruiker de computer een redeneerkracht van de gemiddelde mens geeft [82].

Het bestuderen van onvolledige of onnauwkeurige mentale modellen is ook belangrijk om twee redenen. Ten eerste kunnen de fouten die een leerling maakt helpen met het onthullen van hoe de leerprocessen moeten zijn. Ten tweede kunnen typische onjuiste mentale modellen worden begrepen en kunnen instructeurs en ontwerpers van leermaterialen veranderingen aanbrengen in de leerstof om deze typische fouten te minimaliseren [84].

Een incompleet of verkeerd mentaal model leidt soms tot misverstanden over hoe de code wordt uitgevoerd en kan leiden tot problemen bij 'code tracing'. Leerlingen weten misschien hoe ze een waarde aan een variabele toewijzen, maar weten niet hoe deze variabele en waarde in het computergeheugen worden opgeslagen [81]. Een ander voorbeeld hiervan is dat sommige studenten in het onderzoek van Bayman en Mayer dachten dat de volledige vergelijking "LET A = B + 1" wordt opgeslagen in het computergeheugen [79].

3.1.3 Wiskundige kennis

Wiskundige kennis is een voor- en een nadeel voor informatica. Wiskunde is fundamenteel aanwezig in de informatica, van het leren coderen van wiskundige uitdrukkingen in een programmeertaal tot het converteren van en naar hexadecimale of binaire betaalsystemen [85]. Maar wiskundige kennis kan ook een bron van misconcepties zijn.

Leerlingen zonder bestaande programmeerkennis, die naar een programmeer les komen, kennen het concept variabele zoals deze is uitgelegd en wordt gebruikt in wiskunde op de middelbare school. In de wiskunde is een variabele een naam, meestal een letter uit het alfabet, die staat voor numerieke waarde of andere algebraïsche objecten [86]. In algebra op de middelbare school hoeven leerlingen de variabele niet te declareren voordat ze deze kunnen gebruiken. Dit kan een van de redenen zijn dat studenten vaak de declaraties van variabele in Java vergeten [87].

Een voorbeeld van de verschillen tussen programmeren en rekenen is dat in de talen C en Fortran het resultaat van een deling tussen twee integers, bijvoorbeeld 5/9, nul is. Over de uitkomst van 5/9 zijn veel interpretaties mogelijk. Kinderen op de basisschool kunnen het antwoord geven van 0 rest 5 omdat ze nog denken in gehele getallen. Zodra er meer wiskundige kennis is zal het antwoord 0,55 ook mogelijk zijn. Wanneer leerlingen meer wiskundige ervaring hebben zullen de vaker deze manier van deling gebruiken en dus op het antwoord 0,55 uitkomen. Hierdoor zal het principe van deling in de programmeertalen C en Fortran voor een misconceptie kunnen zorgen [88].

Een ander voorbeeld is dat in de wiskunde een variabele kan staan voor een integer of een heel groot getal of zelfs een getal met oneindig getallen. Hierdoor ontstaat de misconceptie dat studenten denken dat er geen limiet is aan de grootte van de waarden die opgeslagen kunnen worden in een variabele [86]. Een andere misconceptie die wordt meegenomen uit de wiskunde is dat een variabele maar één keer een waarde kan krijgen. Binnen één wiskundig probleem kan de waarde van een variabele x niet 10 in het begin zijn en aan het einde veranderen naar 20 [89].

4. Hedy ontwerp

Hedy neemt een nieuwe benadering aan om beginners te leren programmeren. Hedy is een graduele programmeertaal, wat inhoudt dat je geleidelijk kennis maakt met de programmeerregels en dat de syntax steeds complexer wordt. Hedy leert beginners de syntax aan op een manier welke is gebaseerd op hoe interpunctie wordt aangeleerd bij beginnende lezers van een natuurlijke taal. Het uiteindelijke doel van Hedy is om te beginnen met een kleine en eenvoudige Python-achtige taal waarvan de syntaxregels langzaam en geleidelijk complexer gemaakt wordt, totdat beginners Python zelf onder de knie hebben en deze programmeertaal kunnen gebruiken. De doelgroep van de taal is ongeveer 11 jaar [70].

Hedy is opgebouwd in levels die de niveaus moeten weergeven. In elk level worden er bepaalde voorbeeldopdrachten gegeven waarmee een kind kan leren Hedy te gebruiken. In elk volgende level worden er nieuwe commando's toegevoegd.

In level 1 wordt op de volgende manier gevraagd wat je lievelingskleur is en het antwoord zal weergegeven worden op je scherm.

```
1 ask Wat is je lievelingskleur?
2 echo dus je lievelingskleur is...
```

Figuur 4 Hedy level 1

Level 1 kan voor veel kinderen het eerste moment zijn waarop ze kennismaken met programmeercode. Om deze reden wordt er nog geen gebruik gemaakt van moeilijkere termen of printen tussen aanhalingstekens.

In level 2 wordt het concept 'variabele' geïntroduceerd. In onderstaande voorbeeld wordt dezelfde opdracht gegeven als in level 1 alleen nu wordt er gebruik gemaakt van de nieuwe termen.

```
1 kleur is ask Wat is je lievelingskleur?
2 print Jouw favoriet is dus kleur
```

Figuur 5 Hedy level 2

De naam van de variabele is 'kleur' hierin zal worden opgeslagen door middel van 'is' wat er met 'ask' gevraagd wordt. Er wordt nog enkel gebruik gemaakt van woorden om acties te beschrijven tekens, zoals het '=' teken worden in latere levels geïntroduceerd.

In level 2 kan er een probleem worden ontdekt met het printen van een naam die ook een variabele is. Daarom wordt in level 3 het printen tussen aanhalingstekens toegevoegd.

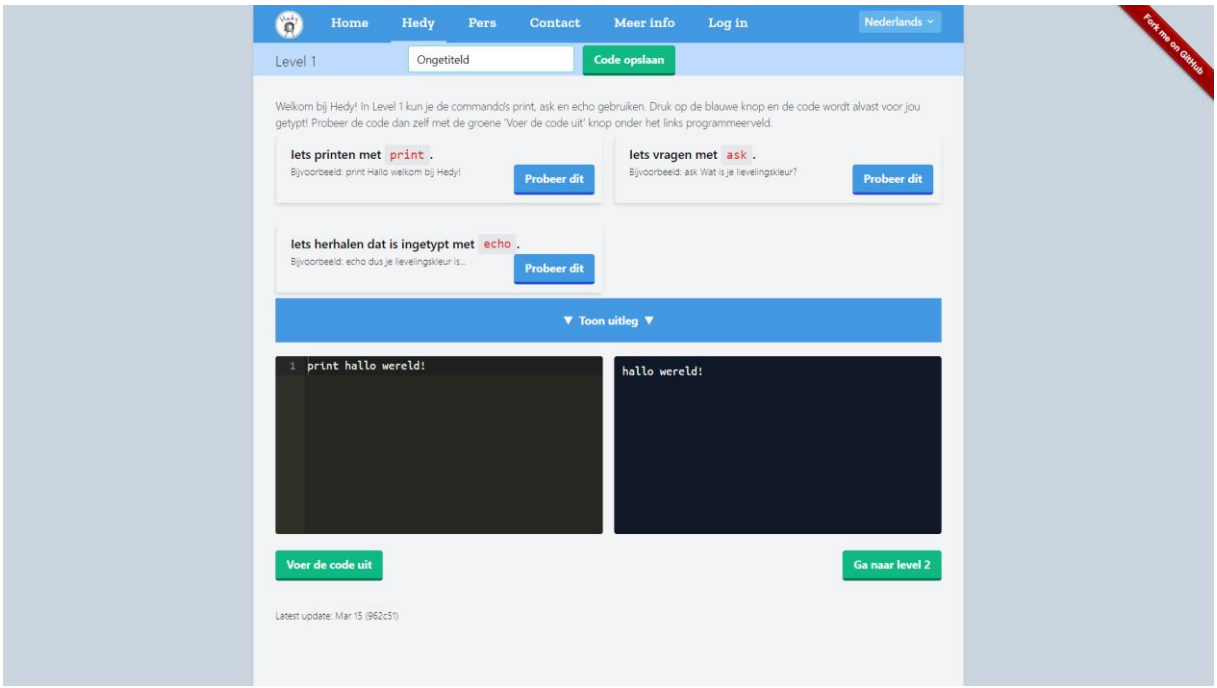
```
1 kleur is ask Wat is je lievelingskleur?
2 print 'Jouw favoriet is dus ' kleur
```

Figuur 6 Hedy level 3

Op deze manier evolueert de opdracht: "je lievelingskleur invoeren en deze weergeven op het scherm" van level 1 tot level 3.

4.1 Gebruikersinterface

De huidige gebruikersinterface van Hedy wordt getoond in Figuur 7. De interface bevat aan de linkerkant een editor waarin je code kunt invoeren en aan de rechterkant een veld voor de uitvoer van deze code. Elk level bevat video's met uitleg over dit level en schriftelijke opdrachten om te oefenen met dit level. Er zijn ook knoppen, gemarkeerd met 'Probeer dit', die de opdrachten welke in dit level zijn geïntroduceerd automatisch in de editor invoert [90].



Figuur 7 Hedy gebruikersinterface

5. Onderzoek

In dit hoofdstuk zullen we een opzet schrijven voor een onderzoek die bedoeld is om antwoord te geven op de vraag: *"Hoe komen bekende programmeermisconcepties voor in de programmeertaal Hedy?".* Het onderzoek zal een kwalitatief onderzoek zijn bedoeld voor kinderen.

5.1 Data verzameling

Voor het verzamelen van de kwalitatieve data maken we gebruik van de think-aloud methode. De think-aloud methode is een methode waarbij deelnemers alle woorden spreken die in hen opkomen bij het voltooien van een taak, zoals het oplossen van een probleem in een programmeertaak. Deze methode is geschikt om achterliggende begrippen en frustraties van deelnemers te identificeren [91].

5.2 Deelnemers

Deze onderzoeksopzet is bedoeld voor leerlingen van de groepen 7 en 8 op de basisschool. Met de criteria voor deelname dat een deelnemer niet beschikt over significante programmeerervaring.

5.2.1 Consent Forms

Alle deelnemers uit de beschreven groepen zullen deelnemen aan de lessen. Er zal echter alleen data verzameld worden van de leerlingen van wie daar expliciet toestemming voor hebben gegeven door middel van het Consentformulier (zie bijlage 7.1)

5.3 Misconcepties

Om te meten welke mogelijke misconcepties kinderen zouden hebben, hebben we een aantal specifieke opdrachten gemaakt voor Hedy welke tot een misconceptie zouden kunnen leiden. We hebben deze programmeermisconcepties geselecteerd uit een lijst met misconcepties van Sorva [92]. Deze lijst van misconcepties is samengesteld op basis van eerdere onderzoeken naar misconcepties bij programmeren, voornamelijk bij volwassenen. Een goed voorbeeld van een misconceptie in de lijst die van toepassing is op Hedy is: *"9: a variable can hold multiple values at the same time/ 'remembers' old values".* In de lijst van Sorva [92] staan ook misconcepties die niet van toepassing zijn op Hedy. In onderstaande tabel staan de misconcepties die we hebben geselecteerd en de reden waarom we voor deze hebben gekozen

Nr	Onderwerp	Beschrijving	Waarom relevant
3	Algemeen	Waarden worden automatisch bijgewerkt volgens een logische context	Ragonis en Ben-Ari ontdekte in 2005 al dat studenten het moeilijk vonden om een algemeen beeld te krijgen van de uitvoering van een programma dat een probleem oplost. Een van de bevindingen was dat studenten denken dat waarden automatisch worden bijgewerkt volgens een logische context [93]. Dit is een interessante misconceptie voor Hedy omdat beginnende programmeurs vaak denken dat een computer snapt wat de programmeur bedoelt [83] en ze dus verwachten dat de waarde wordt bijgewerkt.
4	Algemeen	Het systeem staat geen onredelijke beweringen toe.	Beginnende programmeurs denken dat de computer de intentie van de programmeur snapt en ook foutieve handelingen voorkomt [83], [93]. Dit zorgt voor het idee dat het systeem geen onredelijke beweringen zoals 0 = 1 toestaat. Het is van belang dat we ontdekken of de kinderen die programmeren met Hedy ook deze misconceptie hebben omdat de computer hen niet zal corrigeren bij onredelijke beweringen. Dit zorgt ervoor dat ze zelf de fouten ontdekken en niet ervan uitgaan dat het systeem dit aangeeft.
5	Algemeen	Moeilijkheden om de levensduur van waarden te begrijpen	Het is voor een kind belangrijk om het concept variabele onder de knie te krijgen. Dit is belangrijk zodat ze begrijpen hoe een computer omgaat met variabelen, dus hoelang gaat een variabele mee en wat staat er precies in opgeslagen? In Hedy vormen de verschillende levels nog een extra moeilijkheid voor de levensduur van variabele want wat gebeurt er met een waarde tussen de levels in. Uit eerder onderzoek is gebleken dat onervaren programmeurs soms dachten dat een variabele blijft bestaan totdat de computer wordt uitgezet of de variabele expliciet wordt veranderd [94]. De levels in Hedy kunnen bijdrage aan deze misconceptie omdat het voor kinderen kan lijken dat een variabele die in level 1 wordt aangemaakt ook in level 2 zal bestaan.
9	VarAssign	Een variabele kan meerdere waarden tegelijk bevatten/ ‘onthoudt’ oude waarden	Een ander belangrijk principe van het concept variabele is wat er exact wordt opgeslagen. Uit veel oude onderzoeken [82], [94], [95] is gebleken dat dit een veelvoorkomende misconceptie is. Het is goed om te kijken naar hoe kinderen denken wat er in een variabele wordt opgeslagen en op welke manier.
11 & 12	VarAssign	Primitieve toewijzing werkt in de tegengestelde richting – of – primitieve toewijzing werkt in beide richtingen.	Het toewijzen van waarden aan variabele is een ander knelpunt. Bij het statement A=B is het vaak onduidelijk wat er precies gebeurt. Wordt A hier gelijk gemaakt aan B of B gelijk aan A? [95] Maar wanneer het statement wordt veranderd naar lievelingskleur = oranje dan worden er weinig fouten gemaakt.

			Hedy gebruikt niet het '=' teken maar de het woord 'is' het is goed om te onderzoeken of dit juist helpt met het voorkomen van deze misconceptie of het tegenovergestelde gebeurt. Het toewijzen wordt soms ook in beide richtingen geïnterpreteerd [82]. Dit kan bijvoorbeeld wanneer een kind heeft aangegeven dat lievelingskleur = oranje. Dat er dan de verwachting ontstaat dat wanneer er gevraagd wordt naar oranje dat de computer dan lievelingskleur print.
17	VarAssign	De semantiek in natuurlijke taal van variabele namen bepaalt welke waarde aan welke variabele wordt toegewezen.	Een kind zal de variabelen waarschijnlijk een logische naam toekennen, zoals lievelingsdier voor paard of lievelingskleur voor groen. De toewijzing van de naam 'lievelingskleur' voor een variabele waarin paard wordt opgeslagen zal gek klinken omdat dit een onwaarschijnlijke benaming is. Het is interessant om te zien of kinderen het idee hebben dat de naam van een variabele van belang is voor wat erin wordt opgeslagen.
23	Control	Het begrijpen van de volgorde van statemens is lastig	Het kan onlogisch lijken op welke volgorde statements afgelopen worden door een computer en wat voor invloed een statement heeft voor het verdere verloop van het programma. Bijvoorbeeld als een variabele eerst wordt geprint en vervolgens wordt de waarde aangepast. Wordt dan de aangepaste waarde geprint of de initiële waarde. Dit soort basale vaardigheden zijn belangrijk voordat het kind moeilijkere programmeerstof krijgt. Het is dus goed om te ontdekken of deze misconceptie voorkomt bij kinderen in Hedy.
26	Control	Een 'false' condition beëindigt het programma als er geen verdere vertakking is.	In Hedy wordt de if-statement in level 4 geïntroduceerd. Hier wordt er bijna altijd gebruik gemaakt van of een else-statement of alle verdere vertakkingen worden opgevangen. Hierdoor kan het voor kinderen lastig zijn om te zien wat er gebeurt wanneer er niet aan een if-statement wordt voldaan maar er geen andere optie is. Uit eerdere studies [82], [95] is gebleken dat beginnende programmeurs niet weten wat er gebeurt na een 'false' condition. Hierdoor is het belangrijk dat we dit vroeg te weten komen voor Hedy omdat de if-statement een belangrijk programmeerconcept is.

5.4 Procedure

De lessen worden klassikaal gegeven. De eerste les zal in het teken staan van uitleggen wat programmeren is en waarvoor het gebruikt kan worden. We zullen hiervoor level 1 van Hedy gebruiken als voorbeeld en om kinderen al kennis te laten maken met Hedy. Het doel van deze introductie om de leerlingen het nut van programmeren te laten zien en ze enthousiast te maken voor de komende lessen.

Het lesmateriaal wat gebruikt gaat worden voor de lessen is te vinden onder het kopje 'uitleg' op de website van Hedy. Voor het onderzoek zullen we specifieke opdrachten toevoegen aan het huidige lesmateriaal waarbij we kunnen verwachten dat de leerlingen een misconceptie in Hedy tegenkomt.

Een voorbeeld van een specifieke opdracht is:

- Je wilt een rondje gaan lopen door het bos, voordat je het bos in gaat maak je een *variabele* rugzak aan, dit doe je door rugzak op 0 in te stellen. Zodra je deze hebt aangemaakt ga je op pad.
- Tijdens het lopen zie je een 10 op een boom staan, je slaat deze op in je *variabele* rugzak.
- Na een stukje verder te lopen zie je een 20 op een boom staan deze sla je nu op in je *variabele* rugzak.
- Als je vervolgens het bos uitloopt en je komt weer thuis vraagt je moeder wat de waarde is van je rugzak, wat is jouw antwoord?

Met bovenstaande voorbeeld proberen we te ontdekken of een kind het concept variabele onder de knie heeft. Dus of ze begrijpen welke operaties er worden uitgevoerd op de variabele en wat de uiteindelijke waarde is. De verdere specifieke opdrachten zijn te vinden in bijlage 7.2.

5.5 Hypotheses

Uit voorgaande studies [96], [97] blijkt kinderen het lastig vinden om het concept variabele te leren. Met name het toewijzen van een waarde aan een variabele en de volgorde waarop er operaties op deze variabele worden uitgevoerd wordt lastig gevonden. Ook in Hedy zal een variabele gebruikt gaan worden. Daarom verwachten we dat misconcepties met betrekking tot variabelen ook in Hedy gaan voorkomen, dit zijn misconcepties 9 & 23 uit bovenstaande tabel.

Andere studies laten zien dat mensen denken dat computers op een manier de programmeur begrijpen of snappen [83], [94]. Dit staat in contrast met hoe een computer daadwerkelijk werkt, namelijk letterlijk de regels code uitvoeren. Dit kan voor kinderen extra lastig zijn omdat ze nog minder precies hoeven te zijn dan volwassen. Hierdoor kan misconceptie 3 vaak gaan voorkomen omdat de programmeur kan denken "maar dat snapt de computer toch?".

De namen van variabelen kan een andere voorkomende misconceptie zijn. Het is voor een kind logisch om een betekenisvolle naam aan een variabelen toe te wijzen. Dit kan echter ook suggereren dat deze naam betekenis vol is voor de computer. Het kan nuttig zijn om af en toe onzinnige namen voor variabelen te gebruiken. Onzinnige namen kunnen echter wel verassend zijn als het kind dacht dat het programma werkte omdat het de juist woorden 'begreep'. Misconceptie 17 uit de tabel is daarom heel interessant om te onderzoeken.

6. Toekomstig onderzoek

Een onderzoek uitvoeren met de in hoofdstuk 5 beschreven opzet is een logische richting voor toekomstig onderzoek. De resultaten uit dit onderzoek kunnen aantonen welke misconcepties voorkomen in Hedy. De resultaten kunnen vergeleken worden met eerdere onderzoeken naar Scratch om te kijken of een graduele taal, zoals Hedy, invloed heeft op het aantal misconcepties dat kinderen hebben. Een dergelijk onderzoek kan ook nieuwe misconcepties introduceren die specifiek voor Hedy gelden.

Verder hebben we in dit onderzoek gekeken naar misconcepties in Hedy. Een vervolg hierop zou kunnen kijken naar het verschil van misconcepties in een graduele programmeertaal gefocust op kinderen zoals Hedy en een complexere programmeertaal zijn. Hierbij kan men kijken of Hedy een positieve invloed heeft voor nieuwe jonge programmeurs bij het leren van de programmeersyntax en of er op deze manier minder misconcepties ontstaan.

Een andere richting voor een onderzoek kan zijn om te kijken of Hedy ook helpt bij het verminderen van misconcepties bij nieuwe volwassen programmeurs. Hedy is ontwikkeld om kinderen op een graduele manier kennis te laten maken met de programmeersyntax maar misschien helpt dit ook bij volwassenen om minder syntax fouten te maken. De minder complexe syntax kan ook zorgen voor meer begrip van de concepten waardoor er minder misconcepties kunnen ontstaan.

Bibliografie

[1] J. Bennedsen and M. E. Caspersen, 'Failure rates in introductory programming', *SIGCSE bulletin*, vol. 39, no. 2, pp. 32–36, 2007, doi: 10.1145/1272848.1272879.

[2] M. Piteira and S. R. Haddad, 'Innovate in your program computer class: an approach based on a serious game', in *Proceedings of the 2011 Workshop on Open Source and Design of Communication*, New York, NY, USA, Jul. 2011, pp. 49–54, doi: 10.1145/2016716.2016730.

[3] A. Giannakoulas and S. Xinogalos, 'A pilot study on the effectiveness and acceptance of an educational game for teaching programming concepts to primary school students', *Educ Inf Technol*, vol. 23, no. 5, pp. 2029–2052, Sep. 2018, doi: 10.1007/s10639-018-9702-x.

[4] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman, 'THE FIRST OF THE TOP TEN SIGCSE SYMPOSIUM RESEARCH PAPERS OF THE LAST 50 YEARS: Identifying student misconceptions of programming', *ACM Inroads*, vol. 10, no. 2, pp. 65–69, Apr. 2019, doi: 10.1145/3324900.

[5] 'Software Developers : Occupational Outlook Handbook: U.S. Bureau of Labor Statistics'. https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm#tab-6 (accessed Jan. 11, 2021).

[6] M. Konicek, 'Why demand for software engineers is going to stay high', *Medium*, Oct. 12, 2018. https://medium.com/swlh/why-demand-for-software-engineers-is-going-to-stay-high-5acb789c5015 (accessed Jan. 11, 2021).

[7] 'Commercial off the shelf software shortens complex software lifetime', *Mitopia Technologies*, Aug. 10, 2013. https://mitosystems.com/software-evolution/ (accessed Feb. 08, 2021).

[8] S. M. H. Dehaghani and N. Hajrahimi, 'Which Factors Affect Software Projects Maintenance Cost More?', *Acta Inform Med*, vol. 21, no. 1, pp. 63–66, Mar. 2013, doi: 10.5455/AIM.2012.21.63-66.

[9] S. Papert, *Mindstorms: children, computers, and powerful ideas*. New York, N.Y.: Basic Books, 1980.

[10] C. Kelleher and R. Pausch, 'Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers', *ACM Comput. Surv.*, vol. 37, no. 2, pp. 83–137, Jun. 2005, doi: 10.1145/1089733.1089734.

[11] K. Falkner, R. Vivian, and N. Falkner, 'The Australian Digital Technologies Curriculum: Challenge and Opportunity', *New Zealand*, vol. 148, p. 10, 2014.

[12] N. C. C. Brown, S. Sentance, T. Crick, and S. Humphreys, 'Restart: The Resurgence of Computer Science in UK Schools', *ACM Transactions on Computing Education (TOCE)*, vol. 14, no. 2, pp. 1–22, 2014, doi: 10.1145/2602484.

[13] M. E. Caspersen and P. Nowack, 'Computational thinking and practice: a generic approach to computing in Danish high schools', in *Proceedings of the Fifteenth Australasian Computing Education Conference - Volume 136*, AUS, Jan. 2013, pp. 137–143, Accessed: Jan. 18, 2021. [Online].

[14] T. Bell, 'Establishing a nationwide CS curriculum in New Zealand high schools', *Commun. ACM*, vol. 57, no. 2, pp. 28–30, Feb. 2014, doi: 10.1145/2556937.

[15] The Committee on European Computing Education (CECE), 'Informatics Education in Europe: Are We All In The Same Boat?', Association for Computing Machinery, New York, NY, USA, Technical Report, 2017.

[16] M. E. Caspersen and J. Gal-Ezer, 'Informatics for All Committee', p. 16.

[17] M. Smith, 'Computer Science For All', *whitehouse.gov*, Jan. 30, 2016. https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all (accessed Oct. 27, 2020).

[18] R. Bornat and S. Dehnadi, 'Mental models, Consistency and Programming Aptitude', p. 10, 2008.

[19] Y. Bosse and M. A. Gerosa, 'Why is programming so difficult to learn?: Patterns of Difficulties Related to Programming Learning Mid-Stage', *SIGSOFT Softw. Eng. Notes*, vol. 41, no. 6, pp. 1–6, Jan. 2017, doi: 10.1145/3011286.3011301.

[20] T. Jenkins, ‘On the difficulty of learning to program’.
[21] A. Robins, J. Rountree, and N. Rountree, ‘Learning and Teaching Programming: A Review and Discussion’, *Computer science education*, vol. 13, no. 2, pp. 137–172, 2010, doi: 10.1076/csed.13.2.137.14200.
[22] T. Beaubouef and J. Mason, ‘Why the high attrition rate for computer science students: some thoughts and observations’, *SIGCSE bulletin*, vol. 37, no. 2, pp. 103–106, 2005, doi: 10.1145/1083431.1083474.
[23] S. Bergin and R. Reilly, ‘The influence of motivation and comfort-level on learning to program’, p. 12, 2005.
[24] C. Watson and F. W. B. Li, ‘Failure rates in introductory programming revisited’, in *Proceedings of the 2014 conference on Innovation & technology in computer science education - ITiCSE ’14*, Uppsala, Sweden, 2014, pp. 39–44, doi: 10.1145/2591708.2591749.
[25] ‘Bennedsen: Failure rates in introductory programming - Google Scholar’. https://scholar.google.com/scholar?as_ylo=2017&hl=nl&as_sdt=2005&sciodt=0,5&cites=13018675291962708324&scipsc= (accessed Mar. 09, 2021).
[26] J. Bennedsen and M. E. Caspersen, ‘Failure rates in introductory programming: 12 years later’, *ACM Inroads*, vol. 10, no. 2, pp. 30–36, Apr. 2019, doi: 10.1145/3324888.
[27] A. Luxton-Reilly, *Learning to Program is Easy*. 2016, p. 289.
[28] S. Mohorovicic and V. Strcic, ‘An overview of computer programming teaching methods’, in *Central European Conference on Information and Intelligent Systems*, 2011, p. 47.
[29] K. A. Sarpong, J. K. Arthur, and P. Y. Owusu, *Causes of Failure of Students in Computer Programming Courses: The Teacher – Learner Perspective*. .
[30] F. Kalelioğlu, ‘A new way of teaching programming skills to K-12 students: Code.org’, *Computers in Human Behavior*, vol. 52, pp. 200–210, Nov. 2015, doi: 10.1016/j.chb.2015.05.047.
[31] V. Ramalingam, D. LaBelle, and S. Wiedenbeck, ‘Self-efficacy and mental models in learning to program’, in *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, New York, NY, USA, Jun. 2004, pp. 171–175, doi: 10.1145/1007996.1008042.
[32] ‘“Programmeren en coderen in de klas” vandaag van start’. https://www.nationaleonderwijsgids.nl/cursussen/nieuws/37580-programmeren-en-coderen-in-de-klas-vandaag-van-start.html (accessed Feb. 10, 2021).
[33] ‘Programmeeronderwijs, the next level op Programmeerplein NOT 2019’. https://www.nationaleonderwijsgids.nl/mbo/nieuws/45911-programmeeronderwijs-the-next-level-op-programmeerplein-not-2019.html (accessed Feb. 10, 2021).
[34] R. Pijpers and L. Stiller, ‘Computing-onderwijs (Wat kunnen we leren van de Britten?)’, Kennisnet, Zoetermeer, 2015.
[35] M. Berry, *Computing in the national curriculum - A guide for primary teachers*. Bedford, 2013.
[36] R. Pijpers and L. Stiller, ‘Computing at school - Een jaar na de invoering’, Kennisnet, Zoetermeer, 2016.
[37] Didactief, ‘Digidenkers’, Oktober 2016.
[38] D. Thompson, T. Bell, P. Andreae, and A. Robins, ‘The role of teachers in implementing curriculum changes’, in *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE ’13*, Denver, Colorado, USA, 2013, p. 245, doi: 10.1145/2445196.2445272.
[39] N. Ragonis, O. Hazzan, and J. Gal-Ezer, ‘A survey of computer science teacher preparation programs in Israel tells us: computer science deserves a designated high school teacher preparation!’, Jan. 2010, pp. 401–405, doi: 10.1145/1734263.1734402.
[40] S. Furber, ‘Shut down or restart? The way forward for computing in UK schools.’, The Royal Society, London, 2012.
[41] KNAW, ‘Digitale geletterdheid in het voortgezet onderwijs: Vaardigheden en attitudes voor de 21ste eeuw.’, Koninklijke Nederlandse Akademie van Wetenschappen, Amsterdam, 2012.
[42] J. Voogt and N. P. Roblin, ‘21st Century Skills. Discussienota’, p. 66.

[43] J. Voogt and A. ten Brummelhuis, 'Information Literacy in the Netherlands: Rise, Fall and Revival', in *Reflections on the History of Computers in Education: Early Use of Computers and Teaching about Computing in Schools*, A. Tatnall and B. Davey, Eds. Berlin, Heidelberg: Springer, 2014, pp. 83–93.
[44] A. Thijs, P. Fisser, and M. van der Hoeven, '21e eeuwse vaardigheden in het curriculum van het funderend onderwijs', SLO, Enschede, 2014.
[45] R. Pijpers, 'Handboek digitale geletterdheid 2021/2022', Kennisnet, Zoetermeer, 2021.
[46] 'Samenvatting Digitale geletterdheid – Curriculum.nu'. https://www.curriculum.nu/voorstellen/digitale-geletterdheid/samenvatting-digitale-geletterdheid/ (accessed Feb. 11, 2021).
[47] R. Pijpers, 'Handboek Digitale Geletterdheid 2017/2018', Kennisnet, Zoetermeer, 2017.
[48] 'Uitwerking Digitale geletterdheid – Curriculum.nu'. https://www.curriculum.nu/voorstellen/digitale-geletterdheid/uitwerking-digitale-geletterdheid/ (accessed Feb. 11, 2021).
[49] P. Schnabel, 'Ons Onderwijs 2032: Eindadvies', Platform Onderwijs 2032, Den Haag, rapport, 2016. Accessed: Feb. 11, 2021. [Online]. Available: https://www.rijksoverheid.nl/documenten/rapporten/2016/01/23/eindadvies-platform-onderwijs2032-ons-onderwijs2032.
[50] Stichting Kennisnet, 'Programmeren in het PO', 2016.
[51] J. M. Wing, 'Computational thinking', *Commun. ACM*, vol. 49, no. 3, pp. 33–35, Mar. 2006, doi: 10.1145/1118178.1118215.
[52] I. Lee, S. Grover, F. Martin, S. Pillai, and J. Malyn-Smith, 'Computational Thinking from a Disciplinary Perspective: Integrating Computational Thinking in K-12 Science, Technology, Engineering, and Mathematics Education', *J Sci Educ Technol*, vol. 29, no. 1, pp. 1–8, Feb. 2020, doi: 10.1007/s10956-019-09803-w.
[53] A. Bundy, 'Computational Thinking is Pervasive', p. 3, 2007.
[54] C. M. University and C. M. University, 'Research - Machine Learning - CMU - Carnegie Mellon University', *Machine Learning | Carnegie Mellon University*. https://www.ml.cmu.edu//research/index.html (accessed Feb. 12, 2021).
[55] J. Fisher and T. A. Henzinger, 'Executable cell biology', *Nature Biotechnology*, vol. 25, no. 11, Art. no. 11, Nov. 2007, doi: 10.1038/nbt1356.
[56] D. J. Abraham, A. Blum, and T. Sandholm, 'Clearing Algorithms for Barter Exchange Markets: Enabling Nationwide Kidney Exchanges', p. 10, 2007.
[57] S. Grover and R. Pea, 'Computational Thinking: A Competency Whose Time Has Come', 2017.
[58] J. Lockwood, 'Computational Thinking in Education: Where does it fit?', p. 58.
[59] J. P. Smith III, A. A. diSessa, and J. Roschelle, 'Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition', *The Journal of the learning sciences*, vol. 3, no. 2, pp. 115–163, 1994, doi: 10.1207/s15327809jls0302_1.
[60] A. E. Fleury, 'Parameter passing: the rules the students construct', *SIGCSE Bull.*, vol. 23, no. 1, pp. 283–286, Mar. 1991, doi: 10.1145/107005.107066.
[61] M. Raadt, R. Watson, and M. Toleman, 'Teaching and assessing programming strategies explicitly', *Conferences in Research and Practice in Information Technology Series*, vol. 95, pp. 45–54, 2009.
[62] K. Goldman *et al.*, 'Setting the Scope of Concept Inventories for Introductory Computing Subjects', *ACM Trans. Comput. Educ.*, vol. 10, no. 2, p. 5:1-5:29, Jun. 2010, doi: 10.1145/1789934.1789935.
[63] L. Porter, C. Bailey Lee, and B. Simon, 'Halving fail rates using peer instruction: a study of four computer science courses', in *Proceeding of the 44th ACM technical symposium on Computer science education*, New York, NY, USA, Mar. 2013, pp. 177–182, doi: 10.1145/2445196.2445250.
[64] T. Sirkiä and J. Sorva, 'Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises', in *Proceedings of the 12th Koli Calling International*

Conference on Computing Education Research, New York, NY, USA, Nov. 2012, pp. 19–28, doi: 10.1145/2401796.2401799.
[65] J. Sorva, V. Karavirta, and L. Malmi, 'A Review of Generic Program Visualization Systems for Introductory Programming Education', *ACM Trans. Comput. Educ.*, vol. 13, no. 4, p. 15:1-15:64, Nov. 2013, doi: 10.1145/2490822.
[66] N. C. C. Brown and A. Altadmri, 'Investigating novice programming mistakes: educator beliefs vs. student data', in *Proceedings of the tenth annual conference on International computing education research - ICER '14*, Glasgow, Scotland, United Kingdom, 2014, pp. 43–50, doi: 10.1145/2632320.2632343.
[67] A. Stefik and S. Siebert, 'An Empirical Investigation into Programming Language Syntax', *ACM Transactions on Computing Education (TOCE)*, vol. 13, no. 4, pp. 1–40, 2013, doi: 10.1145/2534973.
[68] S. Garner, P. Haden, and A. Robins, 'My Program is Correct But it Doesn't Run: A Preliminary Investigation of Novice Programmers' Problems', p. 8.
[69] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx, 'Understanding the syntax barrier for novices', in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE '11*, Darmstadt, Germany, 2011, p. 208, doi: 10.1145/1999747.1999807.
[70] F. Hermans, 'Hedy: A Gradual Language for Programming Education', in *Proceedings of the 2020 ACM Conference on International Computing Education Research*, New York, NY, USA, Aug. 2020, pp. 259–270, doi: 10.1145/3372782.3406262.
[71] M. Resnick *et al.*, 'Scratch: programming for all', *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009, doi: 10.1145/1592761.1592779.
[72] K. E. Gray and M. Flatt, 'ProfessorJ: a gradual introduction to Java through language levels', in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, New York, NY, USA, Oct. 2003, pp. 170–177, doi: 10.1145/949344.949394.
[73] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi, 'The TeachScheme! Project: Computing and Programming for Every Student', *Computer Science Education*, vol. 14, no. 1, pp. 55–77, Jan. 2004, doi: 10.1076/csed.14.1.55.23499.
[74] J. Piaget, 'Piaget's Theory', in *Piaget and His School: A Reader in Developmental Psychology*, B. Inhelder, H. H. Chipman, and C. Zwingmann, Eds. Berlin, Heidelberg: Springer, 1976, pp. 11–23.
[75] F. Thompson and S. Logue, 'An Exploration of Common Student Misconceptions in Science', *International Education Journal*, vol. 7, no. 4, pp. 553–559, Sep. 2006.
[76] J. Longfield, 'Discrepant teaching events: Using an inquiry stance to address students' misconceptions', *International Journal of Teaching and Learning in Higher Education*, vol. 21, no. 2, p. 266, 2009.
[77] M. Ben-Ari, 'Constructivism in Computer Science Education', p. 29.
[78] C. Miller, 'Metonymy and reference-point errors in novice programming', *Computer Science Education*, vol. 24, Sep. 2014, doi: 10.1080/08993408.2014.952500.
[79] P. Bayman and R. E. Mayer, 'A diagnosis of beginning programmers' misconceptions of BASIC programming statements', *Commun. ACM*, vol. 26, no. 9, pp. 677–679, Sep. 1983, doi: 10.1145/358172.358408.
[80] J. C. Spohrer and E. Soloway, 'Novice mistakes: are the folk wisdoms correct?', *Commun. ACM*, vol. 29, no. 7, pp. 624–632, Jul. 1986, doi: 10.1145/6138.6145.
[81] L. Ma, 'Investigating and Improving Novice Programmers' Mental Models of Programming Concepts', University of Strathclyde, 2007.
[82] D. Sleeman, R. T. Putnam, J. Baxter, and L. Kuspa, 'Pascal and High School Students: A Study of Errors', *Journal of Educational Computing Research*, vol. 2, no. 1, pp. 5–23, Feb. 1986, doi: 10.2190/2XPP-LTYH-98NQ-BU77.
[83] R. D. Pea, 'Language-independent conceptual "'bugs'" in novice programming', p. 13.

[84] D. Gentner, ‘Mental Models, Psychology of’, in *International Encyclopedia of the Social & Behavioral Sciences*, Elsevier, 2001, pp. 9683–9687.
[85] T. Beaubouef, ‘Why computer science students need math’, *SIGCSE Bull.*, vol. 34, no. 4, pp. 57–59, Dec. 2002, doi: 10.1145/820127.820166.
[86] D. Doukakis, M. Grigoriadou, and G. Tsaganou, ‘Understanding the programming variable concept with animated interactive analogies’, 2007.
[87] J. Jackson, M. Cobb, and C. Carver, ‘Identifying Top Java Errors for Novice Programmers’, 2005, pp. T4C-T4C, doi: 10.1109/FIE.2005.1611967.
[88] A. Ebrahimi, ‘Novice programmer errors: language constructs and plan composition’, *International Journal of Human-Computer Studies*, vol. 41, no. 4, pp. 457–480, 1994, doi: https://doi.org/10.1006/ijhc.1994.1069.
[89] D. Plass, ‘Identifying and Addressing Common Programming Misconceptions with Variables (Part 1)’, p. 66.
[90] ‘Welcome to Hedy!’ https://www.hedycode.com/ (accessed Feb. 27, 2021).
[91] S. Mathison, ‘Encyclopedia of Evaluation’, 0 vols, Feb. 2021, doi: 10.4135/9781412950558.
[92] J. Sorva, ‘Visual Program Simulation in Introductory Programming Education’, 2012.
[93] N. Ragonis and M. Ben-Ari, ‘A long-term investigation of the comprehension of OOP concepts by novices’, *Computer Science Education*, vol. 15, no. 3, pp. 203–221, Sep. 2005, doi: 10.1080/08993400500224310.
[94] B. Du Boulay, ‘Some Difficulties of Learning to Program’, *Journal of Educational Computing Research*, vol. 2, no. 1, pp. 57–73, Feb. 1986, doi: 10.2190/3LFX-9RRF-67T8-UVK9.
[95] R. T. Putnam, D. Sleeman, J. A. Baxter, and L. K. Kuspa, ‘A Summary of Misconceptions of High School Basic Programmers’, *Journal of Educational Computing Research*, vol. 2, no. 4, pp. 459–472, Nov. 1986, doi: 10.2190/FGN9-DJ2F-86V8-3FAU.
[96] A. Swidan, F. Hermans, and M. Smit, ‘Programming Misconceptions for School Students’, in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, New York, NY, USA, Aug. 2018, pp. 151–159, doi: 10.1145/3230977.3230995.
[97] Ž. Žanko, M. Mladenović, and I. Boljat, ‘Misconceptions about variables at the K-12 level’, *Educ Inf Technol*, vol. 24, no. 2, pp. 1251–1268, Mar. 2019, doi: 10.1007/s10639-018-9824-1.

7. Appendix

7.1 Consent form

Universiteit Leiden

Toestemmingsformulier Programmeeronderwijs

Beste ouder(s)/verzorger(s)

In de klas van uw kind zullen programmeerlessen gegeven worden vanuit Programming Education Research Lab (Lab) een onderzoeksinstelling van Universiteit Leiden. Tijdens deze lessen zal er een onderzoek plaatsvinden over misconcepties in de programmeertaal Hedy. Daarom willen wij uw kind vragen om mee te doen aan dit onderzoek.

In dit onderzoek willen we graag meten welke concepten van het programmeeronderwijs wel en niet goed begrepen worden door kinderen. Specifiek zullen we hier kijken naar de programmeertaal Hedy.

Tijdens de lessen kunnen de kinderen aangeven wanneer ze tegen een probleem aan lopen. Vervolgens zal er gevraagd worden naar de gedachtegang achter het probleem en hoe ze tegen dit probleem zijn aangelopen. Deze gedachtegang zal genoteerd worden.

Vertrouwelijkheid
Namen van kinderen zullen niet worden gebruikt in de publicatie van het onderzoek.

De kinderen of de ouder(s)/verzorger(s) mogen te allen tijde aangeven te willen stoppen met het onderzoek.

Toestemming

Ik begrijp de achtergrond van het onderzoek en ik begrijp dat de resultaten geanonimiseerd zullen worden.

Als ouder/verzorger van _____ (naam kind) stem ik in met deelname aan dit onderzoek.

Geboortedatum kind: _____

Groep: _____

Datum: _____

Handtekening: _____

7.2 Specifieke opdrachten

Misconceptie 3: Waarden worden automatisch bijgewerkt volgens een logische context.

Level: 2+

Opdracht:

We gaan nu proberen onze eigen naam op het scherm te krijgen.

Laat de computer aan je vragen, door middel van ask, wat je naam is en sla deze op in een *variabele*. Maak nu gebruik van print en zet je naam op het scherm.

Oplossingen:

Goed: Als het kind neerzet: “naam is ‘Ricardo’ of ‘Felienne’

Fout: Als het kind neerzet: “naam is ‘jouw naam’ of verwacht dat de computer zijn/haar naam weet.

Te verwachten gedrag:

Wat interessant zou zijn is als het inlogsysteem zou werken, dat zou voor kinderen helpen met ‘denken’ dat de computer hun naam al weet

Misconceptie 4: Het systeem staat geen onredelijke beweringen toe.
Misconceptie 17: De semantiek in natuurlijke taal van variabele namen bepaalt welke waarde aan welke variabele wordt toegewezen.

Level: 2

Opdracht:

We willen onszelf gaan voorstellen aan de computer

We willen de computer naast onze naam nog wat informatie geven over onszelf. Zo willen we de computer ons lievelingsdier vertellen en wat we het liefst eten.

Maak gebruik van ask om de computer aan je te laten vragen wat je lievelingsdier is en sla deze op in een *variabele*. Vergeet dit niet uit te printen!

Laat de computer je nogmaals een vraag stellen met ask waarin de computer vraagt wat je favoriete eten is, sla deze ook weer op in een *variabele*. Vergeet deze ook uit te printen.

Als het goed is kan de computer nu zeggen wat je lievelingsdier is en wat je graag eet.

Vraag aan de klas: “Denken jullie dat het voor de computer iets uitmaakt in wat voor ‘naam’(variabele) jullie je lievelingsdier opslaan? Probeer anders deze ‘naam’ in totaal iets anders te veranderen of om te draaien met de ‘naam’ van jullie lievelingseten en kijk wat er gebeurt”

Oplossingen:

Goed: Als er op het scherm 2 antwoorden komen te staan waarvan 1 het lievelingsdier is en 1 het lievelingseten.

Fout: Als het niet lukt om de 2 antwoorden op het scherm te tonen

Als het niet lukt om de antwoorden in *variabelen* op te slaan

Te verwachten gedrag:

Dat ze in het eerste deel van de opdracht hun lievelingseten opslaan in een variabele ‘eten’ en hun lievelingsdier opslaan in een variabele ‘dier’. Na de gestelde vraag zullen veel kinderen gaan spelen met de namen en erachter komen dat de naam niet logisch hoeft te zijn voor de waarde die erin staat.

Dit kan gevolgen hebben voor de namen van variabelen die ze in de volgende Hedy lessen gaan geven.

Misconceptie 5: Moeilijkheden om de levensduur van waarden te begrijpen.

Level: 3+

Opdracht:

We leren langzaam steeds meer kennis maken met variabelen, dit zijn hele handige ‘containertjes’ waar we getallen of woorden in kunnen opslaan. Deze variabelen hebben echter wel een gelimiteerd bereik. Zo worden ze niet opgeslagen tussen levels in.

Als er een les is waar er in meerdere levels geprogrammeerd wordt een opdracht toevoegen dat ze in het eerste level een variabele ‘x’ moeten opslaan met hun schoenmaat en wanneer ze naar het volgende level gaan deze weer moeten opvragen

Oplossingen:

Goed: De variabele is in dit level nooit geïnitialiseerd dus deze bestaat niet.

Fout: Dat de leerling verwacht dat de waarde die in een eerder level in deze variabele stond opgeslagen hier nog steeds in staat.

Te verwachten gedrag:

Als het kind goed bezig is met programmeren en denkt dat de computer de variabelen onthoudt dan zal hij/zij niet denken dat de variabele wordt ‘vergeten’ tussen de levels door. Het kind zal hierdoor de variabele waarschijnlijk gebruiken alsof hij zo ingesteld is als in het vorige level.

Misconceptie 9: Een variabele kan meerdere waarden tegelijk bevatten/ 'onthoudt' oude waarden.

Level: 3+

Opdracht:

Je wilt een rondje gaan lopen door het bos, voordat je het bos in gaat maak je een *variabele* rugzak aan, dit doe je door rugzak op 0 in te stellen. Zodra je deze hebt aangemaakt ga je op pad.

Tijdens het lopen zie je een 10 op een boom staan, je slaat deze op in je *variabele* rugzak.

Na een stukje verder te lopen zie je een 20 op een boom staan deze sla je nu op in je *variabele* rugzak.

Als je vervolgens het bos uitloopt en je komt weer thuis vraagt je moeder wat de waarde is van je rugzak, wat is jouw antwoord?

Oplossingen:

Goed: De waarde van rugzak is 20 (het laatste getal)

Fout: De waarde van rugzak is 10 (het eerste getal)
De waarde van rugzak is 15 (het gemiddelde)
De waarde van rugzak is 30 (de som)

Te verwachten gedrag:

De opdracht is vrij duidelijk, de kinderen (participanten) zullen beginnen met: "Rugzak is 0" waarna ze vervolgens "Rugzak is 10" en "Rugzak is 20" zullen typen. Wat interessant is, is wat de kinderen denken dat er nu in de Rugzak staat. Is het, het eerste getal? Of toch juist het tweede? Of wordt er automatisch iets berekend binnen een variabele?

Misconceptie 11&12: Primitieve toewijzing werkt in tegengestelde richting & primitieve toewijzing werkt in beide richtingen.

Level: onduidelijk is momenteel nog niet mogelijk

Opdracht:

Zoals je langzaam werkt doe je eigenlijk alles door gebruik te maken van variabelen. Zo kan je ook de waarde die in een variabele staat opgeslagen doorgeven aan een andere variabele.

Maak een variabele ‘a’ en sla hier 20 in op. Maak een tweede variabele ‘b’ en sla hier 10 in op. Sla nu de waarde van ‘b’ op in ‘a’. Als we nu beide variabele printen wat is er gebeurt?

Oplossingen:

Goed: A is 20 en B is 20

Fout: A is 10 en B is 20
A is 20 en B is 0
A is ‘B’ en B is 20

Te verwachten gedrag:

Het kind kan denken dat A het getal dat in B staat opgeslagen afpakt, dus op deze manier B 0 maakt. Of kan juist denken dat ze de getallen omwisselen omdat het beide kanten op werkt. Of dat er in A echt de letter B wordt opgeslagen

Misconceptie 23: Het begrijpen van de volgorde van statements is lastig

Level: 7

Opdracht:

Nu we langzaam ook kunnen rekenen met de computer kunnen we oefenen met sommetjes maken. Zo kunnen we nu de plus en minus gebruiken en de tafels oefenen. Ook kunnen we 2 *variabelen* bij elkaar optellen.

Maak een variabele een die je getal1 noemt en sla hier 10 in op. Maak daaronder nog een variabele aan die je getal2 noemt en sla hier 15 in op.

Zorg er nu voor dat deze getallen bij elkaar opgeteld worden in een derde variabele 'totaal'. (let hierop dat ze getal1+getal2 gebruiken en niet de cijfers)

Verander vervolgens de waarde van getal1 in 4 en de waarde van getal2 in 8. Als we nu het totale getal printen, dus variabele totaal wat is de oplossing?

Oplossingen:

Goed: Het goede antwoord is totaal is 25

Fout: Het foute antwoord is totaal is 12

Te verwachten gedrag:

Als ze snappen dat getal1 en getal2 zijn aangepast nadat ze zijn opgeteld dan zullen ze op het goede antwoord uitkomen, we verwachten echter dat kinderen denken dat als de getal waardes veranderen dit overal gebeurt. Dus ook de waardes die eerder in het programma voorkwamen.

Misconceptie 26: Een ‘false’ condition beëindigt het programma als er geen verdere vertakking is.

Level: 7 (of 4+ maar dat is tot level 7 zonder inspringen)

Opdracht:

In dit level maken we ook kennis met de if-statement. Wat gebeurt er als er geen else statement is maar de if-statement ook niet klopt?

In de uitleg staat al een voorbeeld van een programma dat kijkt of mijn zusje stiekem op de computer zat. Als zij erachter zat dan werd er neergezet ‘verboden toegang voor jou’. We gaan opnieuw zo’n test maken maar dan zonder dat ze het weet.

Maak een programma dat vraagt hoe je heet. Wanneer jouw naam wordt getypt dan mag je computeren. Wat gebeurt er als we geen ‘else’ statement toevoegen? Dit houdt in dat wanneer er iets anders dan jouw naam wordt getypt er niets gebeurt.

Als we dan onder deze test nog regels typen wat zal hier mee gebeuren?

Oplossingen:

Goed: De code loopt gewoon door met het statement daaronder

Fout: Het programma stopt
Het programma begint overnieuw

Te verwachten gedrag:

Kinderen hebben tot nu toe enkel geleerd hoe een if/else statement samen werkt. Doordat ze niet weten wat er gebeurt als er enkel een if statement staat kan er van alles gebeuren. Er kan gedacht worden dat het programma stopt met werken, of er opnieuw om de naam wordt gevraagd totdat de ‘goede’ naam wordt ingetypt.