6 februari 2020

VOORKOMEN VAN MISCONCEPTIES BIJ INTRODUCTIE PROGRAMMEREN IN HET VOORGEZET ONDERWIJS

ONDERZOEK VAN ONDERWIJS

René Boschma, s1581899

UNIVERSITY OF TWENTE.

Samenvatting

Een van de onderdelen in het examenprogramma van het schoolvak Informatica in het voortgezet onderwijs is programmeren. Beginnende programmeurs hebben vaak moeite met het aanleren van deze vaardigheid (Mohamad Gobil, Shukor & Mohtar, 2009). Misconcepties kunnen hier een negatieve rol in spelen.

In dit onderzoek probeerden we de volgende vraag te beantwoorden: *"Hoe geef je een lessenserie introductie tot programmeren aan 4 HAVO en 4 VWO die misconcepties voorkomt/verhelpt?"*. Systematisch werden ontwerpeisen in kaart gebracht en uitgewerkt. Vanuit deze ontwerpeisen werd lesmateriaal ontwikkeld waarmee een pilot gedraaid is op een school in het voortgezet onderwijs.

Gekozen is om het lesmateriaal via een online platform aan te bieden. Het lesmateriaal bestaat uit verscheidene opdrachten die al het benodigde materiaal bevatten. Leerlingen maken veel kleine opdrachten om de basis programmeerconcepten te leren. Deze concepten moeten vervolgens worden toegepast op grotere problemen.

Uit dit onderzoek concluderen we dat het beschreven lesmateriaal in de praktijk bruikbaar is om programmeren aan te leren. Met de beschreven lesopbouw en het uitgewerkte lesmateriaal behalen leerlingen de gestelde leerdoelen. Daarnaast werden bij de resultaten van de toets weinig misconcepties gedetecteerd bij de leerlingen. Echter is het onduidelijk of dit door het lesmateriaal komt of dat deze helemaal niet voor zouden komen. Leerlingen vinden het lesmateriaal over het algemeen leuk om mee te werken en geven aan dat ze het gevoel hebben dat ze iets nieuws geleerd hebben.

Inhoudsopgave

1 Inleiding **3**
1.1 Aanleiding . . . 3
1.2 Leeswijzer . . . 3

2 Theoretisch Kader **4**
2.1 Misconcepties . . . 4
2.2 Programmeertaal . . . 4
2.3 Didactiek . . . 4

3 Onderzoeksvragen **6**

4 Methode **7**

5 Fase 1: Vooronderzoek **9**
5.1 Programmeerconcepten . . . 9
5.1.1 Examenprogramma . . . 9
5.1.2 Concretisering Ontwerpeis 1 . . . 10
5.1.3 Concepten . . . 10
5.2 Misconcepties . . . 12
5.2.1 Variabelen . . . 12
5.2.2 Opeenvolging . . . 13
5.2.3 Keuze . . . 13
5.2.4 Herhaling . . . 13
5.2.5 Datatypen . . . 13
5.2.6 Functies . . . 14
5.2.7 Overzicht . . . 15
5.3 Didactiek . . . 15
5.4 Randvoorwaarden . . . 16
5.5 Overzicht ontwerpeisen . . . 16

6 Fase 2: Ontwerp Lesmateriaal **18**
6.1 Uitwerking Ontwerpeisen . . . 18
6.2 Evaluatie . . . 21
6.2.1 Expertpanel . . . 22
6.2.2 Toets . . . 22
6.2.3 Vragenlijst . . . 23

7 Fase 3: Evaluatie **24**
7.1 Opdrachten . . . 24
7.2 Toets . . . 24
7.2.1 Misconcepties . . . 25
7.3 Vragenlijst . . . 28
7.3.1 Planning . . . 28

8 Discussie **29**
8.1 Algemene Bevindingen . . . 31

9 Conclusie **32**

Appendices **35**
A.1 Misconcepties Overzicht . . . 36
A.2 Oefentoets . . . 37
A.3 Vragenlijst . . . 41
A.4 Percentage gemaakte opdrachten per hoofdstuk . . . 49
A.5 Resultaat vragenlijst per hoofdstuk . . . 51
A.6 Opdrachten . . . 52
A.6.1 Printing . . . 52
A.6.2 Variabelen . . . 54
A.6.3 Types en expressies . . . 58
A.6.4 Input en casting . . . 65
A.6.5 Keuze . . . 71
A.6.6 Herhaling . . . 80
A.6.7 Functies . . . 89
A.6.8 Lijsten . . . 97
A.6.9 Opdrachten . . . 104

Hoofdstuk 1

Inleiding

1.1 Aanleiding

Imperatief programmeren is een verplicht onderdeel in het examenprogramma 2018/2019 van HAVO en VWO[1]. Ook in het nieuwe examenprogramma, dat ingaat in het schooljaar 2019, is imperatief programmeren een verplicht onderdeel voor zowel HAVO als VWO[2]. Over het algemeen wordt het geaccepteerd dat het lastig is om te leren programmeren (Bennedsen & Caspersen, 2007). Bennedsen en Caspersen (2019) rapporteren een zakkingspercentage van 28% voor introductie informatica vakken waar programmeren aangeleerd wordt (hogescholen en universiteiten). Daarnaast kennen de resultaten van programmeer vakken veelal een bimodale verdeling (Plass, 2015). Er zijn veel leerlingen die falen en veel leerlingen die hoog scoren met maar weinig ertussenin (Bornat, Dehnadi & Simon, 2008; Corney, 2009; Hudak & Anderson, 1990). Twee verklaringen zijn hiervoor geopperd. De eerste is dat er simpelweg twee soorten leerlingen bestaan, één die wel kan programmeren en één die dit niet kan. Deze theorie, ondanks veel onderzoek, is niet bewezen. De andere verklaring is de Learning Edge Momentum (LEM) hypothese. Deze hypothese bouwt voort op het idee dat leren gebeurt op de randen van wat een mens kent. Op het moment dat voorkennis foutief of niet aanwezig is, belemmert dit het leren van nieuwe concepten. Met andere woorden; het succesvol aanleren van één concept maakt het leren van dicht bij gelegen concepten makkelijker. Bij programmeren, wordt geclaimd, dat concepten dicht bij elkaar liggen. Daarom is het belangrijk om vanaf het begin concepten duidelijk aan te leren om dit effect te voorkomen (Robins, 2010).

Misconcepties kunnen een negatieve rol spelen bij het aanleren van nieuwe concepten. Een programmeer misconceptie is een incorrect beeld van een concept of set van concepten die tot fouten in het lezen en schrijven van code leiden (Sorva, 2009). Door het identificeren van misconcepties kan gericht lesmateriaal ontworpen worden die deze misconcepties aanpakken en het leerproces verbeteren (Peters, 2018). Dit brengt ons op de volgende onderzoeksvraag:

> *Hoe geef je een lessenserie introductie tot programmeren aan 4 HAVO en 4 VWO die misconcepties bij leerlingen tot een minimum beperkt?*

Daarmee is dit onderzoek een ontwerp onderzoek en kent het twee fases: een onderzoeksfase (het theoretische kader) en een ontwerpfase (het daadwerkelijk ontwerpen van het materiaal).

1.2 Leeswijzer

In de rest van dit verslag zullen wij ons onderzoek verder uiteenzetten. Hoofdstuk 2 beschrijft de relevante begrippen en theorieën gevonden in de literatuur. Hoofdstuk 3 beschrijft de onderzoeksvragen opgesteld vanuit het theoretisch kader. Hoofdstuk 4 beschrijft hoe het proces zal gaan lopen om uiteindelijk het lesmateriaal te ontwerpen en hoe het lesmateriaal zal worden geëvalueerd. Hoofdstukken 5, 6 en 7 omvatten het uitgewerkte proces beschreven in Hoofdstuk 4. Hoofdstuk 8 bediscussieert de verkregen resultaten en Hoofdstuk 9 sluit af met een conclusie.

[1] https://www.examenblad.nl/examenstof/informatica-vwo-2/2018/f=/inf_havovwo.pdf
[2] https://www.examenblad.nl/examenstof/informatica-havo-en-vwo-3/2019/f=/examenprogramma_Informatica_havo-vwo.pdf

Hoofdstuk 2

Theoretisch Kader

Voordat relevante onderzoeksvragen opgesteld kunnen worden, dient eerst gekeken te worden welke begrippen en concepten belangrijk zijn. Deze begrippen en concepten zullen vanuit de literatuur geïdentificeerd worden. Dit hoofdstuk zal de gevonden begrippen en concepten beschrijven.

2.1 Misconcepties

Een misconceptie is een incorrect beeld van een concept of set van concepten dat leidt tot het verkeerd toepassen van dit concept of set van concepten. Een misconceptie ontstaat doordat een leerling een foutief mentaal model creëert over een bepaald concept Sorva (2009). Dit foutieve mentale model kan ontstaan door incorrecte aannames of gokken van een leerling. Ook kan dit ontstaan doordat concepten van een domein toegepast worden op een ander domein, terwijl deze niet uitwisselbaar zijn Horstmann (2013).

Misconcepties bij programmeren kunnen bij verschillende concepten optreden. Zo beschrijft Plass (2015) verschillende categorieën misconcepties bij het concept variabelen, Sekiya en Yamaguchi (2013) omtrent het concept herhaling en Kallia en Sentance (2019) betreffend het concept functies.

2.2 Programmeertaal

Om te kunnen programmeren is een taal nodig. Vaak wordt programmeren gedaan door middel van een tekstuele taal. Er wordt dan geprogrammeerd in mens-leesbare code. Tegenwoordig is het mogelijk om met zogeheten "visuele" programmeertalen te programmeren. Deze talen maken gebruik van een zogeheten blokstructuur. In deze talen is een statement (bijvoorbeeld het toewijzen van een variabele) een blok. Deze blokken kunnen op elkaar geplaatst worden om een programma te maken.

Voorbeelden van visuele programmeertalen zijn Scratch en Snap!. Een voordeel van deze talen tegenover tekstuele talen is hun gemakkelijke syntax. Voor een beginnend programmeur kan het lastig zijn syntactische constructen van een taal te leren en te snappen (Hristova, Misra, Rutter & Mercuri, 2003; Spohrer & Soloway, 1986). Meerbaum-Salant, Armoni en Ben-Ari (2013) tonen in hun onderzoek aan dat het mogelijk is om met Scratch programmeerconcepten aan te leren aan middle-school leerlingen (leeftijden tussen 11 en 14 jaar). Verder laten Armoni, Meerbaum-Salant en Ben-Ari (2015) zien dat middle-school leerlingen sneller een tekstuele taal oppakken wanneer ze al eens met Scratch gewerkt hebben. Echter, werd geen significant verschil in de eindresultaten van de experiment en controle groep gevonden.

Voorbeelden van tekstuele programmeertalen die succesvol in high-school onderwijs (leeftijden tussen 15 en 18 jaar) gebruikt zijn, zijn Python, Pascal en Logo (Grandell, Peltomäki, Back & Salakoski, 2006). Van deze talen is Python tot op heden nog steeds een veelgebruikte taal[1].

2.3 Didactiek

Opdrachten zijn cruciaal in het leren programmeren (Hassinen & Mäyrä, 2006). Uit onderzoek van Lahtinen, Ala-Mutka en Järvinen (2006) blijkt dat leerlingen het meeste leren van het zelfstandig werken aan opdrachten. Ook waarderen leerlingen leren door te doen (Staubitz, Klement, Renz, Teusner & Meinel, 2016). Hier is het belangrijk dat opdrachten van het juiste niveau zijn. Te moeilijke opdrachten

[1] https://www.tiobe.com/tiobe-index/

zullen demotiverend werken. Uitdagende en relevante opdrachten zullen motiverend werken (Vihavainen, Paksula & Luukkainen, 2011). Daarnaast is het van belang dat de opdrachten kortetermijndoelen bevatten. Een andere manier om motivatie te verbeteren bij leerlingen is om de leerlingen spelletjes te laten programmeren (Rajaravivarma, 2006). Verder kan een goed ontworpen video instructie de kennis over verschillende aspecten van het programmeren van studenten significant verbeteren (Peters, 2018).

Xie et al. (2019) beschrijft in zijn onderzoek een opbouwende manier van complexiteit om programmeren aan te leren aan studenten. Leerlingen leren eerst code traceren en vervolgens syntax schrijven. In de vervolgstap krijgen leerlingen een programma en moeten ze op een abstract niveau redeneren wat deze doet. Een voorbeeld is hiervan weergegeven in 2.1. Een student krijgt dit programma en moet inzien dat dit programma de waardes van de twee variabelen verwisselt. De laatste stap is dat leerlingen gegeven een probleem, zoals het verwisselen van twee variabelen, zelf een programma schrijven. Edwards et al. (2019) sluit aan bij deze manier van lesopbouw. Weinig instructie is gegeven en leren gaat door het zelfstandig maken van veel opdrachten. De opdrachten hebben een opbouwende moeilijkheid, beginnend bij het vervangen van delen code tot het volledig zelf schrijven van een programma. Bij deze lesopbouw maakten leerlingen naast de opdrachten ook een project. In dit project werd de geleerde theorie toegepast. De leerlingen werkten in tweetallen aan dit project.

```
temp = x
x = y
y = temp
```

Listing 2.1: Voorbeeld code snippet waarbij de waardes van x en y verwisseld worden.

Opdrachten moeten ook gecontroleerd worden op correctheid. Tijdens het oefenen is het daarom van belang dat een leerling zijn of haar code kan uitvoeren om deze op correctheid te testen. Daarnaast is nog het probleem bij opdrachten dat beschrijvingen ambigu kunnen zijn (Staubitz et al., 2016). Hierdoor kan een leerling volgens zijn eigen interpretatie een correct programma gemaakt hebben, maar voldoet deze niet aan de opgestelde eisen van de docent. Om dit laatste te voorkomen is het nodig om nog een extra controle loop in plaats te hebben.

Een manier om ambiguïteit tegen te gaan is door het maken van goede en duidelijke opdrachtomschrijvingen. Hierbij moet een afweging worden gemaakt tussen weinig uitleg, met de kans dat de opdracht verkeerd geïnterpreteerd wordt, of met veel uitleg, met de kans dat leerlingen verdrinken in de hoeveelheid uitleg. Iets wat bij verduidelijking van opdrachten kan helpen is het geven van concrete voorbeelden wat het programma moet produceren (Vihavainen et al., 2011).

Hoofdstuk 3

Onderzoeksvragen

Vanuit het theoretisch kader zijn de volgende zes vragen opgesteld om antwoord te kunnen geven op de onderzoeksvraag: *"Hoe geef je een lessenserie introductie tot programmeren aan 4 HAVO en 4 VWO die misconcepties bij leerlingen tot een minimum beperkt?"*.

A1 Welke programmeerconcepten moeten worden aangeleerd?

A2 Welke programmeer misconcepties zijn er?

A3 Hoe ziet een geschikte didactiek voor het aanleren van programmeerconcepten eruit?

A4 Aan welke randvoorwaarden moet het lesmateriaal voldoen?

B1 Hoe ziet een mogelijke concrete invulling van het lesmateriaal eruit?

C1 Tot hoever voldoet de mogelijke concrete invulling van het lesmateriaal aan de ontwerpeisen?

C2 Tot hoever beheersen de leerlingen de theorie behandeld in het ontworpen lesmateriaal?

De vragen zijn opgedeeld in drie categorieën. Vragen gemarkeerd met een A zijn onderzoekende vragen. Voor deze vragen werd gekeken naar literatuur en de praktijk voor antwoorden. Vragen gemarkeerd met een B zijn vragen die van praktische aard zijn. Voor deze vragen werd gekeken naar antwoorden vanuit de literatuur en de praktijk om een praktische invulling voor het lesmateriaal te vinden. Vragen gemarkeerd met een C zijn evaluatie vragen. Voor deze vragen werd gekeken naar de resultaten van de toets die leerlingen na het maken van het lesmateriaal moesten maken.

Hoofdstuk 4

Methode

Dit hoofdstuk bespreekt de opzet van het onderzoek. Het onderzoek kende drie fases: Fase 1: Vooronderzoek, *Fase 2:* Lesmateriaal ontwikkelen en *Fase 3:* Evaluatie. Wat per fase is uitgevoerd is in dit hoofdstuk uitgewerkt.

Fase 1

In fase 1 werden de onderzoeksvragen gemarkeerd met een A beantwoord. Onderzoeksvragen A1, A2 en A3 zijn beantwoord door middel van een literatuuronderzoek. Onderzoeksvraag A4 is beantwoord door naar de praktijk te kijken en te inventariseren welke limitaties er vanuit de school, waar de pilot gedraaid is, waren. Dit is gedaan door gesprekken aan te gaan met betrokken partijen zoals de informatica docent en de systeembeheerder. Vanuit deze vier onderzoeksvragen zijn ontwerpeisen opgesteld.

Fase 2

In fase 2 is het daadwerkelijke lesmateriaal ontworpen en daarmee antwoord gegeven op onderzoeksvraag B1. Dit ontwerpen werd op basis van de ontwerpeisen, geïdentificeerd in fase 1, gedaan. Fase 2 zelf kent drie stappen. In de eerste stap kregen alle ontwerpeisen een concrete invulling.

In de tweede stap werden de eerste drie hoofdstukken van het materiaal uitgewerkt. Dit materiaal werd vervolgens gestuurd naar 2 informatica docenten (verder te lezen als expertpanel). Eén van deze docenten is de informatica docent op de school waar de pilot uitgevoerd werd. De andere docent is een informatica docent van een andere school. Het doel van deze stap is om vroegtijdig feedback te krijgen op het materiaal. Met het expertpanel probeerden we eventuele ontwerpfouten, op basis van de inzichten van een expert, te identificeren. Op deze manier konden we deze fouten verbeteren, voordat het materiaal aan leerlingen voorgelegd werd. Daarnaast werd de feedback over de eerste drie hoofdstukken meegenomen naar het maken van de rest van de hoofdstukken. Aan het expertpanel werd toelichting over de gekozen lesopbouw gestuurd. Daarnaast werden drie concrete vragen gesteld die verbonden waren aan ontwerpeisen. De concrete vragen zijn uitgewerkt in Hoofdstuk 6.2.1.

In de derde stap van fase 2 werd de feedback van het expertpanel verwerkt en meegenomen om de rest van het lesmateriaal te ontwikkelen. Door tijdgebrek kon voor het uitvoeren van de pilot geen feedback meer worden gevraagd over het gehele materiaal.

Fase 3

Fase 3 geeft antwoord op onderzoeksvragen C1 en C2. Om antwoord te geven op deze onderzoeksvragen is een pilot gedraaid met het ontworpen lesmateriaal uit fase 2. Deze pilot vond plaats op mijn stageschool. Aan deze pilot deden een 4 HAVO en een 4 VWO klas mee bestaande uit respectievelijk 16 en 10 leerlingen. Het ontworpen lesmateriaal behandelt stof die voor deze leerlingen in het curriculum valt. Leerlingen werken tijdens hun normale uren informatica aan het materiaal en doen het niet als iets extra's naast hun normale lessen. Voor deze pilot moesten leerlingen verscheidene opdrachten maken. Daarnaast moesten de leerlingen ook een toets maken en een vragenlijst invullen. De toets en vragenlijst zijn uitgewerkt in Hoofdstuk 6.2.

Naar verscheidene aspecten werd gekeken voor de evaluatie van deze pilot. Ten eerste zijn er de gemaakte opdrachten van de leerlingen. Tijdens de pilot waren de leerling vrij in het kiezen in de hoeveelheid opdrachten die ze maakten. De reden hierachter is dat het lesmateriaal, in onze ogen, meer opdrachten bevat dan strikt noodzakelijk is om een concept te leren. Door de vrijheid kan een sterke leerling sneller door de opdrachten en heeft een zwakkere leerling de optie om meer te oefenen. Voor

het beantwoorden van onderzoeksvraag C1 wordt gekeken naar hoeveel procent van de opdrachten een leerling gemaakt heeft. Verder wordt er ook gekeken naar de hoeveelheid gemaakte opdrachten waarbij een misconceptie behandeld wordt en of deze uiteindelijk nog aanwezig zijn bij een leerling.

Ten tweede is er de toets. Voor de toets zal gekeken worden per vraag of deze goed of fout beantwoord is. Ook zal gekeken worden of er misconcepties schuil gaan in foutieve antwoorden. Dit resultaat zal gebruikt worden om antwoord te geven op onderzoeksvraag C1. Daarnaast zal een cijfer aan de toets gekoppeld worden. Deze zal berekend worden volgens de formule: $cijfer = \frac{punten\ behaald}{totaal\ punten} * 10$. Bij deze cijfers zal gekeken worden naar het klassen gemiddelde en de verdeling. Een klassen gemiddelde boven de 6 en minimaal 60% van de leerlingen een voldoende beschouwen we als goed. Dit resultaat zal gebruikt worden om antwoord te geven op onderzoeksvraag C2.

Ten derde is er de vragenlijst. De vragenlijst heeft als doel om te achterhalen hoe de leerlingen het lesmateriaal hebben ervaren. De resultaten van de vragenlijst zullen gebruikt worden om antwoord te geven op onderzoeksvraag C1.

Op basis van de resultaten uit fase 3 zal gekeken worden of het beschreven lesmateriaal in dit onderzoek een geschikte invulling geeft voor het geven van een lessenserie introductie tot programmeren aan 4 HAVO en 4 VWO die misconcepties bij leerlingen tot een minimum beperkt.

Hoofdstuk 5

Fase 1: Vooronderzoek

5.1 Programmeerconcepten

In 2019 wordt een nieuw examenprogramma voor het schoolvak informatica geïntroduceerd. Hierbij is programmeren (domein D in het examenprogramma) een verplicht onderdeel voor HAVO en VWO. De leerlingen waar deze lessenserie aan gegeven zal worden vallen nog onder het oude examenprogramma. Echter, is het doel dat dit lesmateriaal hergebruikt kan worden. Daarom moet het lesmateriaal ook alvast voorbereiden op het nieuwe examenprogramma. Dit levert de volgende ontwerpeis op:

> *Ontwerpeis 1:* Het lesmateriaal moet voldoen aan het oude examenprogramma en alvast voorbereiden op het nieuwe examenprogramma.

Hieronder is uitgewerkt wat volgens het oude en het nieuwe examenprogramma moet worden aangeleerd.

5.1.1 Examenprogramma

Huidig Examenprogramma

Het huidige examenprogramma informatica kent maar één subdomein dat programmeren betreft. Uit deze beschrijving volgt dat een examenkandidaat HAVO/VWO alleen basisconcepten van programmeren hoeft te kennen.

B3 **Software** De kandidaat beheerst eenvoudige datatypen, programmastructuren en programmeertechnieken.

Examenprogramma 2019

In 2019 gaat een nieuw examenprogramma voor het schoolvak informatica in. Hierbij is programmeren (domein D in het examenprogramma) een verplicht onderdeel voor HAVO en VWO. Het programmeerparadigma dat wordt voorgeschreven is imperatief.

SLO (2018a) deelt het Domein D op in twee subdomeinen. Deze zijn als volgt gedefinieerd:

D1 **Ontwikkelen** De kandidaat kan, voor een gegeven doelstelling, programmacomponenten ontwikkelen in een imperatieve programmeertaal, daarbij programmeertaalconstructies gebruiken die abstractie ondersteunen, en programmacomponenten zodanig structureren dat ze door anderen gemakkelijk te begrijpen en te evalueren zijn.

D2 **Inspecteren en aanpassen** De kandidaat kan structuur en werking van gegeven programmacomponenten uitleggen, en zulke programmacomponenten aanpassen op basis van evaluatie of veranderde eisen.

SLO (2018b) noemt de volgende voorbeeldspecificaties voor deze subdomeinen. De leerling kan:

- werkende programmacomponenten ontwikkelen in een imperatieve programmeertaal naar keuze en daarbij
  - Aanduidingen voor data-objecten (variabelen en constanten) en toewijzing om waarden aan variabelen toe te kennen
  - gebruik maken van de algoritmische bouwstenen opeenvolging, keuze en herhaling implementeren met behulp van controlestructuren in de gekozen programmeertaal

  - datatypen implementeren aan de hand van elementaire datatypen en taalconstructies voor datastructuren
  - gebruik maken van abstractiemechanismen (procedures, methoden, subroutines of functies)
  - doelgericht gebruik maken van mechanismen om de leesbaarheid van een programmacomponent te vergroten (betekenisvolle namen voor dataobjecten, gebruik van procedures en functies, commentaar en suggestieve lay-out)
  - debugging en testen inzetten
- vanuit een algoritme een werkend programma ontwikkelen in de gekozen programmeertaal
- de structuur en werking van een gegeven programmacomponent uitleggen
- een gegeven programmacomponent evalueren aan de hand van de eigenschappen correctheid, efficiëntie, en leesbaarheid.
- een bestaande programmacomponent aanpassen
  - als gevolg van een evaluatie van bijvoorbeeld de correctheid, efficiëntie of leesbaarheid van de programmacomponent
  - als gevolg van een veranderde of uitgebreide doelstelling

5.1.2 Concretisering Ontwerpeis 1

Hieronder zijn de leerdoelen die SLO noemt, uitgewerkt en per onderdeel uitgesplitst en gelabeld. C geeft aan dat het een programmeerconcept is, V een vaardigheid en I een inzicht. Het oude examenprogramma dekt alleen leerdoelen af die aangeduid zijn met een C. Doordat er weinig tijd beschikbaar is voor deze lessenserie en leerlingen nog onder het oude examenprogramma vallen, zal deze lessenserie daarom alleen focussen op de leerdoelen aangeduid met een C.

De leerling kan:

C1 aanduidingen voor variabelen herkennen

C2 variabelen waarden toewijzen

C3 gebruik maken van opeenvolging

C4 gebruik maken van keuze

C5 gebruik maken van herhaling

C6 datatypen implementeren aan de hand van elementaire datatypen

C7 gebruik maken van functies

V1 doelgericht gebruik maken van mechanismen om de leesbaarheid van een programmacomponent te vergroten

V2 debugging inzetten

V3 testen inzetten

V4 vanuit een algoritme een werkend programma ontwikkelen

I1 de werking van een gegeven programmacomponent uitleggen

I2 een gegeven programmacomponent evalueren aan de hand van correctheid

I3 een gegeven programmacomponent evalueren aan de hand van efficiëntie

I4 een gegeven programmacomponent evalueren aan de hand van leesbaarheid

I5 een bestaand programmacomponent aanpassen als gevolg van evaluatie

I6 een bestaand programmacomponent aanpassen als gevolg van veranderde of uitgebreide doelstelling

5.1.3 Concepten

Zoals genoemd in de paragraaf hierboven zal voor dit hoofdstuk alleen gefocust worden op de programmeerconcepten. De programmeerconcepten die hierboven genoemd zijn, zijn hieronder verder uitgewerkt en beschreven wat deze precies inhoudt.

Variabelen

Leerdoelen *C1* en *C2* dekken het gebruik van variabelen af. Een variabele is een label voor een waarde die kan veranderen. Deze waarde kan opgevraagd worden door het programma en gebruikt worden. Ook kan deze waarde veranderen tijdens de uitvoering van het programma.

Opeenvolging

Leerdoel *C3* dekt het gebruik af van opeenvolging. Opeenvolging houdt in dat een programma stap voor stap wordt uitgevoerd.

Keuze

Leerdoel *C4* dekt het gebruik af van keuze. Aan de hand van een conditie kan worden bepaald of bepaalde delen van het programma uitgevoerd worden. Keuze betekent in deze zin het kiezen tussen het wel en niet uitvoeren van een bepaald deel van het programma.

Herhaling

Leerdoel *C5* dekt het gebruik af van herhaling. Met herhaling kan een deel van het programma meerdere malen uitgevoerd worden. Aan de hand van een conditie wordt bepaald of het deel nogmaals uitgevoerd moet worden.

Datatypen

Leerdoel *C6* dekt het gebruik af van datatypen. SLO licht niet toe wat er precies onder de term datatype valt. De enige vermelding die gemaakt wordt, is dat deze geïmplementeerd moet kunnen worden aan de hand van elementaire datatypen. Meerdere definities van deze term datatype bestaan binnen de literatuur. Parnas, Shore en Weiss (1976) noemen 5 definities van datatype die binnen de literatuur gevonden zijn:

> *Syntactisch:* Een datatype wordt gedefinieerd puur door de semantische informatie meegegeven tijdens de declaratie van een variabele.
>
> *Mogelijke Waardes:* Een datatype wordt gedefinieerd door de mogelijke waardes die deze kan aannemen.
>
> *Gedrag:* Een datatype wordt gedefinieerd door de set van operaties op de mogelijke waardes van het datatype.
>
> *Representatie:* Een datatype wordt gedefinieerd door hoe het wordt gerepresenteerd in termen van primitieve datatypes.
>
> *Representatie + gedrag:* Een datatype wordt gedefinieerd door hoe het wordt gerepresenteerd in termen van primitieve datatypes plus de set van operaties op het datatype.

Binnen de gevonden definities kan SLO lijsten bedoelen, maar ook het maken en gebruiken van objecten. Dit laatste valt onder een ander programmeer paradigma (object georiënteerd).

Analyseren van een andere lesmethode binnen de informatica geeft ook geen uitsluitsel. Zo behandelt Fundament[1] alleen het veranderen van en naar verschillende primitieve datatypes (ook wel casting genoemd). In dit onderzoek zullen we dezelfde interpretatie als Fundament volgen, omdat deze al gebruikt wordt bij andere scholen door Nederland. Alleen primitieve datatypes en hoe deze omgezet kunnen worden naar andere primitieve datatypes zal behandelt worden.

Functies

Leerdoel *C7* dekt het gebruik af van functies. Een functie definieert een set van statements die uitgevoerd wordt wanneer aangeroepen. Hiermee wordt abstractie mogelijk. Zo kan iemand een functie definiëren die aangeeft of een getal priem is of niet. Een functie kan daarom gezien worden als een soort commando die de computer uitvoert. Naast abstractie, bevorderen functies ook het hergebruik en leesbaarheid van code.

[1] https://www.instruct.nl/methoden/fundament-informatica/

5.2 Misconcepties

Misconcepties kunnen een negatieve rol spelen bij het aanleren van nieuwe concepten. Door het identificeren van misconcepties kan gericht lesmateriaal ontworpen worden dat deze misconcepties aanpakt en het leerproces verbeterd (Peters, 2018). Dit levert de volgende ontwerpeis op:

> *Ontwerpeis 2:* Het lesmateriaal moet misconcepties voorkomen en verhelpen.

In deze sectie behandelen we per geïdentificeerd programmeerconcept uit sectie 5.1 misconcepties gevonden in de literatuur. Elke concrete misconceptie krijgt een afkorting waaraan later in dit verslag gerefereerd wordt. Verwijzingen bestaan uit een letter en een cijfer. De letter geeft aan bij welk concept de misconceptie hoort (*(V)ariabelen*, *(O)peenvolging*, *(K)euze*, *(H)erhaling*, *(D)atatypen* en *(F)uncties*). In dit hoofdstuk zijn verscheidene code fragmenten opgenomen die een misconceptie illustreren. Om duidelijk te maken waar precies de fout zit in deze voorbeelden, zijn delen van de code rood gemarkeerd.

5.2.1 Variabelen

Een van de basisconcepten van programmeren zijn variabelen. Plass (2015) deelt misconcepties omtrent variabelen in de volgende categorieën in: *wiskundig, menselijke interactie* en de *container analogie*. Als laatste is er nog een laatste catch-all: *misvatting semantiek*.

Wiskundig

De eerste categorie misconcepties komt voort uit de wiskunde. De meeste leerlingen zullen variabelen eerst tegenkomen bij wiskunde. Een som $x + 5 = 10$ bijvoorbeeld heeft als variabele x. De variabelen in de wiskunde zijn anders dan in de informatica (Horstmann, 2013). In de wiskunde beschrijft een algebraïsche expressie de relatie tussen variabelen en getallen. Deze moet vervolgens opgelost worden (*V1*).

Code snippet 5.1 laat een code voorbeeld zien waar deze misconceptie op kan treden. Een leerling kan denken dat $x = y + 3$ opgelost moet worden. Dit heeft als resultaat dat y de waarde 2 krijgt. In werkelijkheid is dit niet zo en blijft de waarde van y onveranderd. Ook hoort bij deze categorie de misconceptie dat variabelen maar eenmaal een waarde toegewezen kunnen krijgen (*V2)*.

```
int x = 5;
int y;
x = y + 3;
```

Listing 5.1: Voorbeeld code snippet in Java die misconceptie V1 demonstreert

Naast wiskundige gerelateerde fouten hebben beginnende programmeurs ook moeite met het correct uitvoeren van wiskundige operaties (Mohamad Gobil et al., 2009).

Menselijke Interactie

Deze categorie misconceptie komt voort uit de verwachtingen die we voor een computer hebben. Beginnende programmeurs kunnen programmeren zien als ”praten” met de computer en verwachten dat computers begrijpen wat hun code betekent (vergelijkbaar met het praten met mensen (Pea, Soloway & C., 1987)). Zo wordt gedacht dat een variabele met de naam *kleinste* niet het grootste getal kan bevatten. Of wanneer een leerling een variabele maakt met de naam *kleinste* deze automatisch de kleinste waarde zal bevatten (*V3*).

Container Analogie

Deze categorie misconceptie komt voort uit de container analogie die vaak gebruikt wordt om variabelen uit te leggen (Ben-Ari, 1998). Een variabele is in deze analogie gelijk aan een doos. Deze doos kun je een naam geven, zodat je deze weer terug kan vinden. Ook kun je een waarde in de doos stoppen en er vervolgens weer uithalen. Deze analogie werkt tot op zekere hoogte, maar gaat bijvoorbeeld mis wanneer de waarde van doos A naar doos B verplaatst wordt. In deze analogie zou nu de inhoud van doos A leeg zijn (*V4*). In een programma is dit niet het geval en wordt de waarde gekopieerd. Ook is het mogelijk dat leerlingen denken dat er meerdere waardes bij een variabele kunnen horen, net zoals je meerdere spullen in een doos kunt stoppen (*V5*).

Misvatting Semantiek

De laatste categorie misconceptie over variabelen dekt misvatting rondom de semantiek van toewijs statements af. Deze misconcepties zijn (Ma, Ferguson, Roper & Wood, 2007):

V6 Toewijzing van links naar rechts (i.p.v. van rechts naar links)

V7 De $=$ als vergelijk operator zien, i.p.v. een toewijs operator

V8 Bij een toewijzing de waardes van twee variabelen verwisseld worden

V9 Resultaten kunnen alleen opgeslagen worden in variabelen niet genoemd in de expressie ($a = a + 1$)

5.2.2 Opeenvolging

Bij het concept opeenvolging kon geen concrete misconceptie geïdentificeerd worden vanuit de literatuur. Ook zijn er geen moeilijkheden gevonden.

5.2.3 Keuze

Bij het concept keuze kon geen concrete misconceptie geïdentificeerd worden vanuit de literatuur. Wel zijn er moeilijkheden gevonden bij dit concept. Mohamad Gobil et al. (2009) merkt op dat studenten moeite hebben met het syntactisch correct schrijven van een if-statement. Ook gebruiken studenten een rijtje verschillende if statements in plaats van een enkele if-then-else structuur.

5.2.4 Herhaling

Sekiya en Yamaguchi (2013) hebben 7 misconcepties gevonden omtrent de combinatie van herhaling en keuze. In het onderzoek kregen leerlingen de opdracht de output van een programma te geven. De foute antwoorden zijn geanalyseerd. Hierbij is gekeken naar welke fout tijdens het traceren is opgetreden. Deze fouten zijn opgesplitst in verschillende type misconcepties. Vijf van de zeven misconcepties hebben te maken met het aanzien van een variabele voor een andere. Een voorbeeld hiervan is weergegeven in Listing 5.2. Het originele programma is weergegeven in dit voorbeeld. De rode *a* betreft de verkeerd gelezen variabele. De fout die leerlingen gemaakt hebben is dat ze de *a* als een een *i* gezien hebben. Studenten kregen achteraf het nagekeken werk terug met de vraag hoe ze tot het gegeven antwoord gekomen waren. Studenten droegen als reden voor dit foutieve antwoord aan dat ze de *a* als een *i* gelezen hadden. Gezien de gegeven verklaringen van deze leerlingen, zullen we deze misconcepties buiten beschouwing laten tijdens dit onderzoek.

```
def a3(a)
  ans = 0
  for i in 1..a
    ans = ans + a
  end
  p ans
end
```

Listing 5.2: Voorbeeld code fragment in Ruby waarbij leerlingen aan moesten geven welke waarde uiteindelijk geprint werd. De waarde die door de leerlingen gegeven werd, correspondeerde met het vervangen van de onderstreepte *a* met een *i* (de control variabele)

De volgende misconceptie betreft wanneer in een loop de control variabele (in Listing 5.2 variabele *i*) niet gebruikt wordt, deze maar 1 keer uitgevoerd wordt (alsof de for loop er niet stond) (*H1*). De 7e misconceptie geïdentificeerd betreft dat voorgaande waardes, van een variabele die aangepast wordt in een loop, bewaard blijven (*H2*). Hiermee wordt de variabele in de loop als een soort array gezien.

Nog een misconceptie geïdentificeerd door Hermans, Swidan en Smit (2018) is dat de uitvoering van de body meteen stopt wanneer de control conditie false wordt binnen de body van de herhaling (*H3*).

5.2.5 Datatypen

Bij het concept datatypen kon geen concrete misconceptie geïdentificeerd worden vanuit de literatuur. Wel zijn er moeilijkheden gevonden bij dit concept. Mohamad Gobil et al. (2009) vond dat studenten moeite hebben met het gebruik maken van het juiste datatype. Een voorbeeld van wat vaak mis ging was het resultaat van een integer deling. Hierbij denken studenten dat de uitkomst een decimaal getal oplevert, terwijl dit juist een geheel getal oplevert.

5.2.6 Functies

Kallia en Sentance (2019) identificeerden zeven verschillende misconcepties. Bij de eerste misconceptie denken studenten dat de return statement gebruikt moet worden na het aanroepen van de functie om de waarde van de functie te printen en terug te geven terwijl dit niet nodig is(*F1*). Een voorbeeld hiervan is weergegeven in Listing 5.3. Hierbij moet volgens de student de *return* statement na het aanroepen van de functie staan.

```
def uitvoer(cijfer):
  if cijfer > 5:
    resultaat = "voldoende"
  else:
    resultaat = "onvoldoende"

def main():
  print(uitvoer(6))
  return resultaat
```

Listing 5.3: Voorbeeld code fragment in Python waarbij leerlingen de return statement gebruiken na het aanroepen van de functie om de waarde te printen en terug te geven terwijl dit niet nodig is (*F1*).

Bij de tweede misconceptie denken studenten dat wanneer een functie een variabele teruggeeft, deze dezelfde naam moet hebben als de functie zelf (*F2*). *Bijvoorbeeld:* de functie *oppervlakte* moet de variabele met naam *oppervlakte* teruggeven (een andere naam mag niet). Bij de derde misconceptie denken studenten dat parameters binnen de functie een waarde toegewezen moeten krijgen om gebruikt te kunnen worden (*F3*). Een voorbeeld hiervan is weergegeven in Listing 5.4. Zonder de toewijzing van parameter *getal* zou volgens een student deze variabele niet gebruikt kunnen worden.

```
def verdubbel(getal):
  getal = 10
  return getal*2
```

Listing 5.4: Voorbeeld code fragment in Python waarbij leerlingen denken dat de parameter een waarde toegewezen moet krijgen om gebruikt te kunnen worden (*F4*). Zonder de toewijzing zou volgens een student de variabele *parameter* niet gebruikt kunnen worden.

Bij de vierde misconceptie denken studenten dat parameters en argumenten dezelfde naam moeten hebben (*F4*). Een voorbeeld hiervan is weergegeven in Listing 5.5. In dit voorbeeld moet, volgens de misconceptie, de parameter van de functie *doe_niks* dezelfde naam hebben als het gebruikte argument (in dit geval *waarde*).

```
def doe_niks( waarde ):
  return waarde

def main():
  waarde = 42
  print(doe_niks( waarde ))
```

Listing 5.5: Voorbeeld code fragment in Python waarbij leerlingen denken dat parameters en argumenten dezelfde naam moeten hebben (*F4*). In dit voorbeeld moet, volgens de misconceptie, de parameter van de functie *doe_niks* dezelfde naam hebben als de gebruikte variabele (in dit geval *waarde*)

Bij de vijfde misconceptie denken studenten dat argumenten ook de return waarde moeten bevatten (*F5*). Een voorbeeld hiervan is weergegeven in Listing 5.6. In dit voorbeeld wordt de variabele *resultaat* toegevoegd aan de lijst van argumenten. Dit is niet correct.

```
def som(n1, n2):
  resultaat = n1+n2
  return resultaat

def main():
    getal1 = 10
    getal2 = 5
    print(som(getal1, getal2, resultaat))
```

Listing 5.6: Voorbeeld code fragment in Python waarbij studenten denken dat argumenten ook de return waarde moeten bevatten (*F5*). In dit voorbeeld wordt de variabele *resultaat* toegevoegd aan de lijst van argumenten.

Bij de zesde misconceptie denken studenten dat locale variabelen van functie tot functie benadert kunnen worden. Een student denkt, bij deze misconceptie, dat de variabele gedefinieerd in de scope van een functie benadert kan worden door een andere functie. Bij de zevende misconceptie denken studenten dat het aanroepen van een functie automatisch het resultaat weergeeft op het scherm. In dat geval hoeft volgens de student de *print* functie niet meer uitgevoerd te worden.

5.2.7 Overzicht

In dit hoofdstuk zijn verschillende misconcepties per onderwerp geïdentificeerd voor ontwerpeis 2. Een overzicht van deze misconcepties is hieronder te vinden.

V1 Oplossen wiskundige formule

V2 Variabele kan maar 1 maal toegewezen worden

V3 Variabele naam heeft invloed op waarde van variabele

V4 Variabele toewijzen aan andere variabele zorgt ervoor dat eerste variabele zijn waarde kwijtraakt.

V5 Variabele kan meerdere waardes tegelijk bevatten

V6 Toewijzing van links naar rechts

V7 De = als vergelijking operator zien

V8 Bij een toewijzing de waardes van twee variabelen verwisseld worden

V9 Resultaten kunnen alleen opgeslagen worden in een variabele niet genoemd in de expressie

H1 Wanneer in een loop de control variabele niet gebruikt wordt, dan wordt de loop maar 1 keer uitgevoerd

H2 Voorgaande waardes van een variabele in een loop blijven bewaard

H3 Wanneer een control variabele false wordt binnen de body van de loop, stopt de loop meteen met uitvoeren.

F1 Return statement moet gebruikt worden na de aanroep van de functie om de waarde van de functie te printen en terug te geven.

F2 Return variabele moet dezelfde naam hebben als de naam van de functie

F3 Parameters moeten expliciet gedefinieerd worden om gebruikt te kunnen worden binnen een functie

F4 Parameters en argumenten moeten dezelfde naam hebben

F5 Argumenten moeten ook de return waarde bevatten

F6 Locale variabelen kunnen van functie tot functie benaderd worden

F7 Het aanroepen van een functie geeft het resultaat weer

5.3 Didactiek

Om programmeren aan te leren dient een geschikte didactische vorm gevonden te worden. Opdrachten zijn hierbij cruciaal (Hassinen & Mäyrä, 2006). Vihavainen et al. (2011) stelt dat uitdagende en relevante opdrachten met kortetermijndoelen motivatie verbeteren bij leerlingen. Dit levert de volgende ontwerpeis op:

> *Ontwerpeis 3:* Het lesmateriaal moet uitdagende en relevante opdrachten met kortetermijndoelen bevatten.

Xie et al. (2019) pleit ervoor dat programmeren het beste aangeleerd kan worden door leerlingen eerst te laten lezen en vervolgens pas te laten schrijven. Edwards et al. (2019) sluit aan bij deze manier van lesopbouw; opdrachten hebben een opbouwende moeilijkheid, beginnend bij het vervangen van delen code tot het volledig zelf schrijven van een programma. Dit levert de volgende ontwerpeis op:

> *Ontwerpeis 4:* Het lesmateriaal moet een opbouwende structuur kennen waarbij leerlingen zelfstandig eerst leren code te lezen, vervolgens aanpassen waarna ze uiteindelijk zelf code schrijven.

Om de lesopbouw van Edwards et al. te kunnen uitvoeren is het nodig dat leerlingen direct weten of ze een opdracht correct hebben uitgevoerd. Om te controleren of het programma aan hun eigen verwachtingen voldoet, is het nodig dat leerlingen zelf de code uit kunnen voeren. Dit levert de volgende ontwerpeis op:

> *Ontwerpeis 5:* Het lesmateriaal moet het mogelijk maken om direct code uit te kunnen voeren.

Door ambiguïteit kan een leerling een ander beeld hebben bij een opdracht dan de docent had voorgenomen (Staubitz et al., 2016). Een leerling kan dan volgens zijn of haar eigen interpretatie de opdracht goed gemaakt hebben, terwijl dit niet het geval is. Een tweede vorm van controle, niet door de leerling zelf, kan dit helpen voorkomen. Automatische testen biedt hier een uitkomst. Bij automatisch testen wordt de code uitgevoerd aan de hand van een aantal vooraf gedefinieerde situaties om te bepalen of deze aan de eisen voldoet. Daarnaast bieden deze automatische testen een kortere feedback cycle dan wanneer de docent de code moet nakijken. Dit levert de volgende ontwerpeis op:

Ontwerpeis 6: De opdrachten van het lesmateriaal moeten automatisch gecontroleerd worden.

In de opdracht beschrijving moet een afweging worden gemaakt tussen weinig uitleg, met de kans dat de opdracht verkeerd geïnterpreteerd wordt, of met veel uitleg, met de kans dat leerlingen verdrinken in de hoeveelheid uitleg. Iets wat bij verduidelijking van opdrachten kan helpen is het geven van concrete voorbeelden wat het programma moet produceren (Vihavainen et al., 2011). Dit levert de volgende ontwerpeis op:

Ontwerpeis 7: Het lesmateriaal moet bestaan uit opdrachten die duidelijk en ondubbelzinnig zijn en waar mogelijk voorbeelden bevatten.

Voor programmeren is een programmeertaal nodig. Voorbeelden van tekstuele programmeertalen die succesvol in high-school onderwijs (leeftijden tussen 15 en 18 jaar) gebruikt zijn, zijn Python, Pascal en Logo (Grandell et al., 2006). Dit levert de volgende ontwerpeis op:

Ontwerpeis 8: Het lesmateriaal moet een programmeertaal gebruiken die geschikt is voor leerlingen in het voortgezet onderwijs.

Wij vinden dat het voortgezet onderwijs leerlingen moet voorbereiden op hun latere leven. Hier vinden we het belangrijk dat leerlingen iets leren wat ze later gemakkelijk kunnen toepassen. Het helpt hier als leerlingen niet eerst een andere taal hoeven te leren. Dit levert de volgende ontwerpeis op:

Ontwerpeis 9: Het lesmateriaal moet een tot op heden nog steeds veelgebruikte programmeertaal gebruiken waar leerlingen later mogelijk ook zelf een project mee kunnen starten.

5.4 Randvoorwaarden

Het ontworpen lesmateriaal zal als pilot draaien op een school in het voortgezet onderwijs. Vanuit deze school zijn een aantal randvoorwaarden waaraan voldaan moet worden.

Voor deze lessenserie staan 6 lesweken waarin leerlingen 2 maal een les van 45 minuten hebben. Na deze 6 weken is het schooljaar afgelopen. Dit levert de volgende ontwerpeis op:

Ontwerpeis 10: Het lesmateriaal moet geschikt zijn om gegeven te kunnen worden in 12 lessen à 45 minuut per les.

Op mijn stageschool wordt het onderwerp programmeren gegeven in de vierde klassen. Dit gebeurt zowel voor HAVO als voor VWO. Daarom moet het lesmateriaal geschikt zijn voor zowel 4 HAVO als 4 VWO:

Ontwerpeis 11: Het lesmateriaal moet geschikt zijn voor leerlingen van 4 HAVO en 4 VWO.

Om beter te kunnen leren moeten leerlingen het materiaal ook leuk en interessant vinden om mee te werken:

Ontwerpeis 12: Het lesmateriaal moet voor de leerlingen leuk zijn om mee te werken.

Binnen de klas zijn er grote verschillen tussen leerlingen. Sommige leerlingen hebben al een beetje ervaring met programmeren en/of pakken technische onderwerpen een stuk sneller op dan anderen. Daarom is het nodig dat er differentiatie mogelijk is:

Ontwerpeis 13: Het lesmateriaal moet differentiatie faciliteren.

Het is lastig om tijdig nieuwe software op computers te installeren. Daarom is het gewenst dat het lesmateriaal aangeboden kan worden zonder verdere installatie.

Ontwerpeis 14: Het lesmateriaal moet geen verdere installatie vereisen op de school.

5.5 Overzicht ontwerpeisen

Een overzicht waar alle ontwerpeisen op een rijtje genoemd staan is hieronder weergegeven.

Ontwerpeis 1: Het lesmateriaal moet voldoen aan het oude examenprogramma en alvast voorbereiden op het nieuwe examenprogramma.

Ontwerpeis 2: Het lesmateriaal moet misconcepties voorkomen en verhelpen.

Ontwerpeis 3: Het lesmateriaal moet uitdagende en relevante opdrachten met kortetermijndoelen bevatten.

Ontwerpeis 4: Het lesmateriaal moet een opbouwende structuur kennen waarbij leerlingen zelfstandig eerst leren code te lezen, vervolgens aanpassen waarna ze uiteindelijk zelf code schrijven.

Ontwerpeis 5: Het lesmateriaal moet het mogelijk maken om direct code uit te kunnen voeren.

Ontwerpeis 6: De opdrachten van het lesmateriaal moeten automatisch gecontroleerd worden.

Ontwerpeis 7: Het lesmateriaal moet bestaan uit opdrachten die duidelijk en ondubbelzinnig zijn.

Ontwerpeis 8: Het lesmateriaal moet een programmeertaal gebruiken die geschikt is voor leerlingen in het voortgezet onderwijs.

Ontwerpeis 9: Het lesmateriaal moet een tot op heden nog steeds veelgebruikte programmeertaal gebruiken waar leerlingen later mogelijk ook zelf een project mee kunnen starten.

Ontwerpeis 10: Het lesmateriaal moet geschikt zijn om gegeven te kunnen worden in 12 lessen à 45 minuut per les.

Ontwerpeis 11: Het lesmateriaal moet geschikt zijn voor leerlingen van 4 HAVO en 4 VWO.

Ontwerpeis 12: Het lesmateriaal moet leuk zijn om mee te werken.

Ontwerpeis 13: Het lesmateriaal moet differentiatie faciliteren.

Ontwerpeis 14: Het lesmateriaal moet geen verdere installatie vereisen op de school.

Hoofdstuk 6

Fase 2: Ontwerp Lesmateriaal

In het voorgaande hoofdstuk zijn verscheidene ontwerpeisen genoemd. In dit hoofdstuk wordt uiteengezet hoe op basis van deze ontwerpeisen lesmateriaal ontworpen is. Dit ontwerpen is in drie stappen gedaan. In de eerste stap is gekeken naar welke ontwerpeisen concreet gemaakt moesten worden voordat het materiaal ontworpen kon worden. In de tweede stap is een klein deel van het materiaal uitgewerkt. Dit materiaal is gestuurd naar het expertpanel. In de derde stap is de feedback van het expertpanel verwerkt en meegenomen voor het verder ontwikkelen van het lesmateriaal. Dit materiaal heeft vervolgens als pilot gedraaid op mijn stageschool.

6.1 Uitwerking Ontwerpeisen

Ontwerpeis 1: Het lesmateriaal moet voldoen aan het oude examenprogramma en alvast voorbereiden op het nieuwe examenprogramma. In Hoofdstuk 5.1 zijn verscheidene concepten van het oude en nieuwe examenprogramma in kaart gebracht. Gekozen is om deze concepten onder te verdelen in hoofdstukken. Een overzicht van welke concepten bij welk hoofdstuk aan bod komen is weergegeven in Tabel 6.1. In hoofdstuk 0 wordt uitgelegd wat een programma is en hoe het iets kan weergeven. Ook wordt hierin uitgelegd aan de leerlingen hoe het lesmateriaal opgebouwd is en hoe de onlineomgeving werkt. In hoofdstuk 1 worden variabelen uitgelegd en hoe je de waarde van een variabele kan gebruiken, veranderen en weergeven. In hoofdstuk 2 wordt uitgelegd dat een variabele altijd een type heeft en dat er verschillende operaties bestaan op deze types. In hoofdstuk 3 wordt uitgelegd hoe input van een gebruiker gevraagd kan worden en hoe je het type van een variabele kan veranderen (casten). In hoofdstuk 4 wordt keuze uitgelegd. Hier komen de if-, else- en elif-statements aan bod. In hoofdstuk 5 wordt herhaling uitgelegd. Hierin worden de while- en for-loop uitgelegd. In hoofdstuk 6 worden functies uitgelegd. Hierbij hoort hoe je een functie aanroept en definieert. Ook parameters, argumenten, scoping en return waardes worden hierin behandeld. In hoofdstuk 7 worden lijsten uitgelegd. Dit is geen onderdeel van het examenprogramma, maar dient voor verdieping. Hoofdstuk 8 bevat alleen opdrachten. Deze is bedoeld voor leerlingen die extra uitdaging willen. De opdrachten die hierin stapsgewijs uitgevoerd moeten worden zijn: programmeer de Ceasar-Cipher en het spelletje; Torens van Hanoi.

nr.	Hoofdstuk	Leerdoel
0	Printing	C3
1	Variabelen	C1 C2
2	Types & Operaties	C1 C2 C6
3	Input & Casting	C6
4	Keuze	C4
5	Herhaling	C5
6	Functies	C7
7	Lijsten	
8	Extra Opdrachten	

Tabel 6.1: Verdeling leerdoelen per hoofdstuk.

Ontwerpeis 2: Het lesmateriaal moet misconcepties voorkomen en verhelpen. Om misconcepties te voorkomen en te verhelpen is er voor gekozen om opdrachten toe te voegen waar misconcepties in behandelt worden (Triglianos et al., 2017). Gekozen is om met name bij leesopdrachten een misconceptie te toetsen. De reden hiervoor is dat bij leesopdrachten het gemakkelijker is om opdrachten te verzinnen die specifiek een misconceptie testen dan bij aanpas- en schrijfopdrachten. Een overzicht van welke misconceptie bij welke opdracht behandeld wordt, is weergegeven in Tabel 6.2. In deze tabel

is *H2* (voorgaande waardes van een variabele in een loop blijven bewaard) niet opgenomen, omdat geen geschikte opdracht hiervoor verzonnen kon worden. *F3* (parameters moeten expliciet gedefinieerd worden om gebruikt te kunnen worden binnen een functie) en *F7* (Het aanroepen van een functie geeft het resultaat weer) worden niet specifiek bij opdrachten behandelt, maar komen indirect bij meerdere opdrachten aan bod. Verder valt op dat *V6* (toewijzing van links naar rechts) en *V7* (de = als vergelijking operator zien) geen opdrachten hebben. Tijdens ontwerpen van het lesmateriaal zijn deze misconcepties over het hoofd gezien.

	V1	V2	V3	V4	V6	V7	V8	V9	H1	H3	F1	F2	F4	F5	F6
1.0.0		X													
1.0.3			X												
1.0.4		X		X			X								
1.0.5		X		X			X								
1.0.6			X												
2.0.1	X														
2.0.5		X						X							
5.0.4									X	X					
5.0.6									X	X					
5.0.8										X					
5.0.9										X					
6.0.0											X				
6.0.1												X			
6.0.2											X				
6.0.3													X		
6.0.8												X			
6.0.9												X			
6.0.13															X
6.0.14												X			
6.1.0														X	

Tabel 6.2: Overzicht misconcepties (horizontaal) per opdracht in het lesmateriaal (verticaal). In deze tabel is *H2* niet opgenomen, omdat geen geschikte opdracht hiervoor verzonnen kon worden. *F3* en *F7* worden niet specifiek bij opdrachten behandelt, maar komen indirect bij meerdere opdrachten aan bod.

Ontwerpeis 3: Het lesmateriaal moet uitdagende en relevante opdrachten met kortetermijndoelen bevatten. Het lesmateriaal bestaat uit vele kleine opdrachten. Een leerling zou binnen een les meerdere opdrachten kunnen voltooien. Een voorbeeld van een veelvoorkomende opdracht is dat een leerling een programma van maximaal 10 regels gegeven krijgt en hiervan de uiteindelijke waarde van een variabele moet geven. Ook moeten leerlingen zelf programmaatjes schrijven die een voor hun relevante/herkenbare context bevatten. Een voorbeeld hiervan is een programma dat strafregels schrijft voor de leerling.

Ontwerpeis 4: Het lesmateriaal moet een opbouwende structuur kennen waarbij leerlingen zelfstandig eerst leren code te lezen, vervolgens aanpassen waarna ze uiteindelijk zelf code schrijven. De opbouw van het lesmateriaal volgt een combinatie van Xie et al. (2019) en Edwards et al. (2019). In de lessen zullen weinig tot geen mondelinge instructie plaatsvinden. Leerlingen leren door het maken van opdrachten. Dit maken zal individueel plaatsvinden. Een leerling werkt hoofdstuk per hoofdstuk door waarin de genoemde structuur van opbouwende complexiteit in zit. Elk hoofdstuk bestaat uit drie secties waar opdrachten van een bepaald type worden aangeboden. In de eerste sectie zitten uitsluitend lees opdrachten (wat is de uiteindelijke waarde van het gegeven programma?). De tweede sectie bevat uitsluitend aanpas opdrachten (gegeven is een incorrect/incompleet programma en een doel wat dit programma moet bereiken. Pas het gegeven programma aan, zodat het gestelde doel behaald wordt). De derde sectie bevat uitsluitend schrijfopdrachten (gegeven een probleem, schrijf een programma voor dit probleem). Deze structuur valt terug te vinden in de nummering van de opdrachten. Een opdracht heeft altijd 3 getallen. Het eerste getal geeft het hoofdstuk aan, het tweede getal welk soort opdracht (0=lees, 1=aanpas of 2=schrijf) en het derde getal geeft de hoeveelste opdracht binnen het soort opdracht aan. *Voorbeeld:* Opdracht [1.2.0] is de eerste schrijfopdracht binnen hoofdstuk 1. De nummering van opdrachten beginnen bij 0. Een overzicht van alle opdrachten is te vinden in Appendix A.6.

Ontwerpeis 5: Het lesmateriaal moet het mogelijk maken om direct code uit te kunnen voeren. *Zie ontwerpeis 14.*

Ontwerpeis 6: De opdrachten van het lesmateriaal moeten automatisch gecontroleerd worden. *Zie ontwerpeis 14.*

Ontwerpeis 7: Het lesmateriaal moet bestaan uit opdrachten die duidelijk en ondubbelzinnig zijn. Tijdens het ontwerpen van de opdrachten is er getracht opdrachten zo duidelijk mogelijk uit te leggen. Hierbij is waar mogelijk een voorbeeld gegeven van wat het programma zou moeten weergeven.

Ontwerpeis 8: Het lesmateriaal moet een programmeertaal gebruiken die geschikt is voor leerlingen in het voortgezet onderwijs. Gezien syntactische constructen van sommige talen lastig voor leerlingen kunnen zijn, moet gekozen worden voor een taal die deze constructen op een heldere manier definieert. Het is mogelijk om te kiezen voor een visuele programmeertaal. Uit onze eigen ervaring stellen we dat visuele programmeertalen beperkter zijn in uitdrukkingsvermogen dan tekstuele talen. Dit betekent dat in onze optiek een visuele programmeertaal beter niet gekozen kan worden. Echter, is uit onderzoek gebleken dat leerlingen een concept sneller oppakken bij een tekstuele taal wanneer deze een concept eerder hebben gezien bij een visuele programmeertaal (Armoni et al., 2015). Daarom is gekozen voor een mix van beide. Gekozen is om de lessenserie te laten beginnen met het schrijven van programma's met een visuele programmeertaal. Hour of Code[1] heeft opdrachten beschikbaar vanaf second-grade (7-8 jaar). In deze opdrachten worden programmeerconcepten zoals opeenvolging, keuze en herhaling uitgelegd en toegepast op een visuele manier. Op deze manier maken leerlingen alvast kennis met de programmeerconcepten op een speelse manier. Gekozen is om de Hour of Code variant van Minecraft te volgen[2]. Deze sluit aan op de leefwereld van veel van de leerlingen en behandelt de meeste programmeerconcepten die geïdentificeerd zijn in Hoofdstuk 5.1. Na de eerste les zal overgestapt worden naar een tekstuele taal.

Ontwerpeis 9: Het lesmateriaal moet een tot op heden nog steeds veelgebruikte programmeertaal gebruiken waar leerlingen later ook zelf een project mee kunnen starten. Voorbeelden van tekstuele programmeertalen waarvan de geschiktheid in high-school onderwijs is aangetoond zijn Python, Pascal en Logo (Grandell et al., 2006). Van deze talen is Python gekozen om aan te leren. Python is tot op heden nog steeds een veelgebruikte taal[3]. Daarnaast staat Python bekend om zijn leesbare syntax[4].

Ontwerpeis 10: Het lesmateriaal moet geschikt zijn om gegeven te kunnen worden in 12 lessen à 45 minuut per les. Om vooraf duidelijk te stellen of de leerdoelen in de genoemde tijd aangeleerd kunnen worden, is een planning opgesteld. Deze planning is opgesteld aan de hand van de hoeveelheid lessen die de externe docent van het expertpanel per concept besteedt. Deze docent gebruikt een ander materiaal dan beschreven in dit onderzoek. De planning is weergegeven in Tabel 6.3.

	In de les	Na de les
Les 1	intro + start H1	H1 af
Les 2	start H2	H2 af
Les 3	start H3	H3 af
Les 4	start H4	
Les 5		H4 af
Les 6	start H5	
Les 7		H5 af
Les 8	start H6	
Les 9		
Les 10		H6 af
Les 11	start H7	
Les 12		H7 af

Tabel 6.3: Planning lesmateriaal

Ontwerpeis 11: Het lesmateriaal moet geschikt zijn voor leerlingen van 4 HAVO en 4 VWO. Getracht is niveau en woordgebruik geschikt te maken voor 4 HAVO en 4 VWO leerlingen. Zo zijn zinnen zo kort mogelijk en zijn moeilijke woorden zoveel mogelijk vermeden.

[1] https://hourofcode.com/nl/learn
[2] https://code.org/minecraft
[3] https://www.tiobe.com/tiobe-index/
[4] https://www.python.org/dev/peps/pep-0020/

Ontwerpeis 12: Het lesmateriaal moet leuk zijn om mee te werken. Allereerst waarderen leerlingen leren door te doen (Staubitz et al., 2016). Daarom is een lesopbouw gekozen waarin leerlingen vrijwel alleen leren door te doen. Verder is getracht opdrachten te verzinnen die relevant zijn voor leerlingen. Ook zijn er een aantal opdrachten waar leerlingen spelletjes moeten programmeren (Vihavainen et al., 2011).

Ontwerpeis 13: Het lesmateriaal moet differentiatie faciliteren. Leerlingen maken de opdrachten zelfstandig en bepalen zelf hoeveel opdrachten ze per opdracht maken. Verder is gekozen om geen klassikale uitleg te geven, omdat dit voor belemmering kan zorgen. Zo kan het voorkomen dat leerlingen gaan wachten totdat ze bepaalde uitleg gehad hebben voordat ze er zelf mee aan de slag gaan. Daarnaast is er extra hoofdstuk toegevoegd die extra en moeilijkere opdrachten bevatten. Hierdoor faciliteert het lesmateriaal differentiatie zowel in tempo als niveau.

Ontwerpeis 14: Het lesmateriaal moet geen verdere installatie vereisen op de school. De gemakkelijkste oplossing waar geen verdere installatie voor nodig is, is een software-as-a-service solution. Dit betekent dat het lesmateriaal via de webbrowser toegankelijk is. Verder stelt ontwerpeis 5 dat dit platform direct code moet kunnen uitvoeren. Ook moet dit platform volgens ontwerpeis 6 automatische testen ondersteunen. Daarnaast moet dit platform de gekozen programmeertaal Python ondersteunen.

Er zijn verschillende soorten web-based editors die toestaan code direct via de browser uit te voeren (Tang, Rixner & Warren, 2014). Voorbeelden hiervan zijn JSFiddle[5], CodeSandBox[6] en CodeSkulptor2[7]. Platformen die ook toestaan om opdrachten te definiëren en automatisch te laten testen zijn Codesters[8] en Repl.it Classrooms[9]. Hiervan is de tweede gratis te gebruiken en ondersteunt het tevens Python. Omdat geen ander geschikt alternatief gevonden kon worden, is gekozen om Repl.it te gebruiken.

Repl.it biedt de mogelijkheid om classrooms aan te maken. Binnen deze classrooms kunnen opdrachten worden gedefinieerd waar instructies van de docent bij geplaatst kunnen worden. Hier kan ook uitleg worden weergegeven. Daarnaast is er ook de mogelijkheid voor een docent om per opdracht automatische testen toe te voegen. Wanneer een leerling een opdracht inlevert, zullen de automatische testen, die bij die opdracht horen, worden uitgevoerd. Een leerling kan meteen zien of deze de opdracht correct heeft gemaakt.

6.2 Evaluatie

In de methode is benoemd hoe de pilot geëvalueerd zal gaan worden. Nu ontwerpeisen concreet in beeld zijn gebracht, kan de validatie per ontwerpeis aangegeven worden. Aan de meeste ontwerpeisen werd voldaan tijdens het ontwerpen van het lesmateriaal. Bijvoorbeeld: ontwerpeis 4 stelt dat een veelgebruikte programmeertaal gebruikt moet worden. Echter, geldt dit niet voor alle ontwerpeisen. Voor deze ontwerpeisen zijn andere methodes nodig om te valideren. De drietal methodes die gebruikt zijn om deze te valideren zijn: een expert review, een eindtoets en een questionnaire. Een overzicht van alle methodes en welke methode gebruikt wordt om welke ontwerpeis te valideren, is weergegeven in Tabel 6.4.

[5] https://jsfiddle.net/
[6] https://codesandbox.io/
[7] https://py3.codeskulptor.org/
[8] https://www.codesters.com
[9] https://repl.it/site/classrooms

	design	expertpanel	toets	questionnaire
1	X			
2			X	
3	X			X
4	X			
5	X			
6	X			
7		X		X
8	X			
9	X			
10*		X		
11		X		X
12				X
13	X			
14	X			

Tabel 6.4: Overzicht welke ontwerpeis (verticaal) op welke manier (horizontaal) gevalideerd wordt. **Achteraf zal gekeken worden of de initiële planning werkelijkheid geworden is.*

6.2.1 Expertpanel

H0, H1 en H2 zijn uitgewerkt als draft. Deze draft is gestuurd naar het expertpanel met de volgende vragen:

1. Zijn de opdrachten begrijpelijk, is het duidelijk wat er gevraagd wordt te maken?
2. Is het niveau goed voor 4 HAVO en 4 VWO leerlingen (niet te moeilijk niet te makkelijk)?
3. Zullen de gestelde leerdoelen op deze manier behaald worden door de leerlingen?

Resultaat

Het expertpanel vond dat de opdrachten begrijpelijk geformuleerd waren. Een aantal spellingfouten waren aanwezig en formuleringen konden korter. Het niveau is goed volgens het panel en leerdoelen zullen behaald worden. Eén van het expert panelleden suggereerde dat klassikale uitleg bij bepaalde onderdelen zeer waardevol kan zijn voor sommige leerlingen. Zeker gezien sommige theorie stukken redelijk wat tekst bevatten. Daarnaast werd gesuggereerd om een tempo aan te geven voor leerlingen.

Van de feedback zijn de tekstuele opmerkingen opgenomen en verbeterd. Daarnaast is er een planning voor de leerlingen gemaakt, waarin ze kunnen kijken hoever ze horen te zijn. Expliciet is er voor gekozen om geen klassikale uitleg te geven. De reden hiervoor is dat leerlingen op eigen tempo de opdrachten doorwerken. Dat betekent dat leerlingen tegen iets aan kunnen lopen, terwijl anderen daar nog helemaal niet aan toe zijn gekomen. Wel is er om deze opmerking voor gekozen om elke les met 1 vraag te starten: *"Zijn er nog vragen/onduidelijkheden over opdrachten of concepten waar iemand uitleg over wil?"*. Op het moment dat meerdere leerlingen uitleg willen over hetzelfde, zal het alsnog klassikaal uitgelegd worden. Wanneer een enkele leerling uitleg wil, zal dit persoonlijk gedaan worden.

6.2.2 Toets

Om te verifiëren of leerdoelen behaald zijn en te kijken of misconcepties aanwezig zijn bij leerlingen, is een toets ontworpen. De toets zelf valt te vinden in Appendix A.2. Deze toets test theorie uit H0 tot en met H6 en doet dat door verscheidene bekende misconcepties te toetsen. Een overzicht van welk leerdoel bij welke vraag aan bod komt, is weergegeven in Tabel 6.5. Bij deze toets is getracht vragen zo te ontwerpen dat deze de misconcepties aan het licht brengen. Wegens gebrek aan tijd is ervoor gekozen om alleen traceer vragen te stellen, zodat deze in een deel van de laatste les afgenomen kon worden.

	C1	C2	C3	C4	C5	C6	C7
1	X	X	X				
2	X	X					
3	X	X					
4	X	X	X				
5			X			X	
6				X			
7				X			
8					X		
9					X		
10							X
11							X
12							X
13							X

Tabel 6.5: Overzicht getoetste leerdoelen (horizontaal) per vraag van de toets (verticaal).

Een overzicht van welke vraag welke misconcepties probeert te detecteren is weergegeven in Tabel 6.6. Hier zijn misconcepties *V5* (variabele kan meerdere waardes tegelijk bevatten), *H2* (voorgaande waardes van een variabele in een loop blijven bewaard) en *F3* (parameters moeten expliciet gedefinieerd worden om gebruikt te kunnen worden binnen een functie) niet in opgenomen, omdat er geen geschikte vraag verzonnen kon worden om deze misconcepties te detecteren.

	V1	V2	V3	V4	V6	V7	V8	V9	H1	H3	F1	F2	F4	F5	F6	F7
1	X															
2		X														
3			X													
4				X	X		X									
5																
6						X										
7																
8								X		X						
9								X	X							
10											X					
11																X
12												X	X	X		
13															X	

Tabel 6.6: Overzicht van alle misconcepties (horizontaal) die getoetst worden per vraag in de toets (verticaal). Misconcepties *V5*, *H2* en *F3* zijn niet opgenomen in deze tabel, omdat geen geschikte vraag verzonnen kon worden om deze misconcepties te detecteren.

6.2.3 Vragenlijst

Naast de toets, werd er ook een vragenlijst afgenomen bij de leerlingen. De vragenlijst is gemaakt in Google Forms en is anoniem ingevuld. De vragenlijst is te vinden in Appendix A.3. De vragenlijst bestaat uit drie delen. In het eerste deel wordt naar de voorgaande programmeerervaring van de leerling gevraagd. In het tweede deel wordt gevraagd wat de leerling in het algemeen van het ontworpen lesmateriaal vond. In het derde deel wordt gevraagd wat de leerling per hoofdstuk vond. Daarnaast was er ook ruimte voor de leerlingen om overige opmerkingen achter te laten.

Hoofdstuk 7

Fase 3: Evaluatie

Tijdens het uitvoeren van de pilot is verscheidene data verzameld. Data is afkomstig van de hoeveelheid opdrachten die de leerlingen gemaakt hebben op het online platform Repl.it, een toets en een vragenlijst. In dit hoofdstuk is deze data en de verwerking ervan uiteengezet.

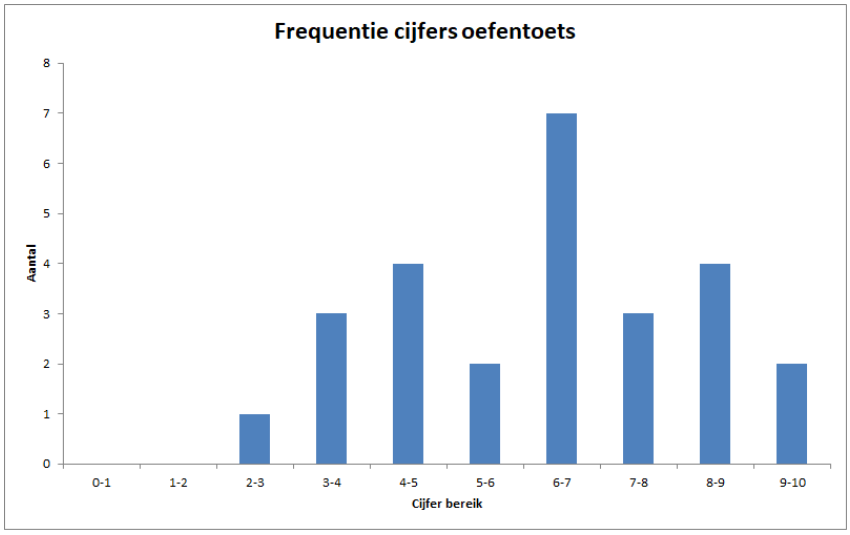
7.1 Opdrachten

Leerlingen waren vrij om te kiezen hoeveel en welke opdrachten ze maakten. In Appendix A.4 is de voortgang per leerling per hoofdstuk weergegeven. Hier valt op dat bijna alle leerlingen in het begin alle opdrachten hebben gemaakt en pas later enkele opdrachten hebben overgeslagen. Het aantal overgeslagen opdrachten neemt bij latere hoofdstukken toe. Daarnaast valt op dat met name de aanpasopdrachten bij latere hoofdstukken overgeslagen worden.

Volgens het Programma van Toetsing en Afsluiting (PTA) moest dit onderwerp afgesloten worden met een praktische opdracht. Daarom is besloten om het daadwerkelijke cijfer dat leerlingen kregen voor dit onderdeel te koppelen aan de hoeveelheid gemaakte schrijfopdrachten (waarbij verschillende formules voor HAVO en VWO zijn gebruikt). Gesteld werd aan de leerlingen dat steeds de eerste twee schrijfopdrachten per hoofdstuk van hoofdstuk 0 tot en met 6 verplicht waren voor een voldoende. Van de 26 leerlingen hebben twee deze eis niet gehaald (leerling 1 en 7).

7.2 Toets

De toets bestond uit 13 vragen en moest in 12 minuten gemaakt worden. Alle leerlingen waren klaar met de toets in 12 minuten. In totaal konden 110 punten gehaald worden. Gemiddeld heeft een leerling een 6.2 gehaald. Van de 26 leerlingen hadden 16 leerlingen (62% van de leerlingen) een voldoende (≥ 6) en 10 een onvoldoende. In Figuur 7.1 is de verdeling van de cijfers weergegeven. Het valt hier op dat deze geen bi-modal verdeling kent. Als gekeken wordt naar de combinatie tussen gemaakte opdrachten en het cijfer voor de toets valt op dat de twee leerlingen die de toets het beste gemaakt hebben (leerlingen 14 en 25) ook de meeste schrijfopdrachten hebben gemaakt.



Figuur 7.1: Verdeling cijfers oefentoets

Tabel 7.1 geeft weer welke leerling welke vraag fout had. In deze tabel staat een X voor een vraag foutief beantwoord. Ook geeft deze per vraag aan hoeveel procent van de leerlingen de vraag fout hadden. Verder geeft deze tabel ook het cijfer weer wat een leerling voor de toets behaald heeft. In deze tabel valt te zien dat de eerste 5 vragen relatief weinig fouten bevatten in vergelijking met de rest van de vragen. Vragen 6, 7, 9, 12 en 13 zijn slecht (>50% fout) gemaakt. Deze vragen bevatten kennis die in hoofdstukken 4 en verder behandelt worden. In Appendix A.4 valt te zien dat de gemaakte opdrachten vanaf hoofdstuk 4 sterk begint af te nemen. Daarnaast zijn vragen 12 en 13 erg slecht (>75% fout) gemaakt. De theorie voor deze vragen wordt behandeld in hoofdstuk 6. Hoofdstuk 6 is in vergelijking met de voorgaande hoofdstukken significant minder gemaakt. Voor leesopdrachten 57.1% tegenover 91.8%, voor aanpasopdrachten 37.2% tegenover 87.6% en schrijfopdrachten 40.8% tegenover 88.8%.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**cijfer**
1						x			x			x	x	6.4
2						x	x					x	x	6.4
3					x		x		x			x	x	5.5
4				x			x		x			x		6.8
5				x		x	x		x		x	x	x	4.1
6						x	x	x	x		x	x	x	3.6
7						x	x	x	x	x	x	x	x	2.7
8						x			x					8.2
9							x	x	x	x	x	x	x	3.6
10		x		x		x	x		x		x	x	x	3.6
11						x	x	x	x			x	x	4.5
12							x					x	x	7.3
13							x		x			x	x	6.4
14													x	9.1
15									x	x				8.2
16									x	x		x	x	6.4
17				x		x						x		7.7
18						x	x		x			x	x	5.5
19							x					x		8.2
20				x		x			x	x	x	x	x	4.1
21							x		x			x	x	6.4
22									x			x	x	7.3
23		x		x		x	x		x			x	x	4.5
24						x	x					x	x	6.4
25													x	9.1
26						x						x		8.2
	0%	8%	0%	23%	4%	54%	62%	15%	69%	19%	23%	85%	77%	6.2

Tabel 7.1: Vragen van de toets (horizontaal) fout per leerling (verticaal). Een X geeft aan dat een leerling de vraag fout had. De laatste kolom geeft het behaalde cijfer van de toets weer. De laatste rij geeft het percentage leerlingen dat de vraag fout beantwoord heeft weer en het gemiddelde cijfer dat behaald is voor de toets.

7.2.1 Misconcepties

Bij het nakijken van de oefentoets is bij een fout antwoord gekeken of dit antwoord gerelateerd was aan een misconceptie. Een overzicht van de gevonden misconcepties is weergegeven in Tabel 7.2. In deze tabel staan op de verticale as de leerlingen en de horizontale as de misconcepties die bij de toets getoetst werden. Een rode kleur geeft aan dat een misconceptie is gedetecteerd bij een antwoord op de toets. Daarnaast geeft deze tabel aan hoeveel procent van de opdrachten een leerling van het lesmateriaal, waarbij een misconceptie aan bod kwam, gemaakt heeft. Verder is een *nvt* ingevuld bij een misconceptie wanneer deze niet in het lesmateriaal opgenomen is of niet gekoppeld is aan enkele opdrachten. De onderste twee rijen geven het gemiddelde aantal aanwezige misconcepties en het gemiddelde gemaakte opdrachten, waarbij een misconceptie behandeld werd, aan. In deze tabel vallen meerdere dingen op. Allereerst valt het op dat V7 (de = als vergelijking operator zien) veel aanwezig is (bij 58% van de leerlingen) en F6 (lokale variabelen kunnen van functie tot functie benadert worden) redelijk aanwezig is (bij 27% van de leerlingen). De rest van de misconcepties zijn bij minder dan 23% van de leerlingen gedetecteerd. Van de 16 misconcepties die in de toets verwerkt waren, zijn er 13 niet gedetecteerd tijdens de toets. Van de 21 leerlingen waarbij een misconceptie gedetecteerd is, hebben 11 meer dan 1 misconceptie. Gemiddeld per leerling zijn er 1.4 misconcepties gedetecteerd.

Ten tweede valt het op dat er een verband lijkt te zijn tussen de gemiddelde aanwezige misconcep-

ties en de gemaakte opdrachten waar deze misconceptie behandeld wordt. Een hoog percentage gemaakte opdrachten heeft vaak een laag gemiddelde aanwezige misconcepties en vice versa. Echter, gaat dit niet altijd op. Zo zijn maar 35% van de opdrachten gemaakt waarbij *F5* behandeld werd en is deze misconceptie niet tijdens de toets gedetecteerd.

	V1	V2	V3	V4	V6	V7	V8	V9	H1	H3	F1	F2	F4	F5	F6	F7
1	100%	100%	100%	100%	nvt	**nvt**	100%	100%	0%	0%	100%	25%	100%	0%	0%	nvt
2	100%	100%	100%	100%	nvt	**nvt**	100%	100%	100%	100%	100%	25%	0%	0%	**0%**	nvt
3	100%	100%	100%	100%	nvt	nvt	100%	100%	**50%**	25%	50%	25%	100%	100%	100%	nvt
4	100%	100%	100%	100%	**nvt**	**nvt**	100%	100%	100%	100%	100%	100%	100%	100%	100%	nvt
5	100%	100%	100%	100%	**nvt**	**nvt**	100%	100%	**0%**	25%	100%	50%	0%	0%	**0%**	nvt
6	100%	75%	100%	100%	nvt	**nvt**	100%	0%	**0%**	0%	100%	75%	100%	0%	0%	nvt
7	100%	100%	100%	100%	nvt	**nvt**	100%	100%	100%	100%	0%	0%	0%	0%	0%	nvt
8	100%	100%	100%	100%	nvt	**nvt**	100%	100%	0%	50%	100%	25%	100%	0%	0%	nvt
9	100%	100%	100%	100%	nvt	nvt	100%	100%	**100%**	**100%**	0%	0%	0%	0%	0%	nvt
10	100%	100%	100%	100%	nvt	**nvt**	100%	100%	**0%**	0%	100%	25%	0%	0%	0%	nvt
11	100%	100%	100%	100%	nvt	**nvt**	100%	100%	**50%**	**75%**	100%	25%	0%	0%	0%	nvt
12	100%	100%	100%	100%	nvt	nvt	100%	100%	100%	100%	100%	50%	100%	100%	100%	nvt
13	100%	100%	100%	100%	nvt	nvt	100%	100%	100%	100%	100%	100%	100%	100%	100%	nvt
14	100%	100%	100%	100%	nvt	nvt	100%	100%	100%	100%	100%	100%	100%	100%	**100%**	nvt
15	100%	100%	100%	100%	nvt	nvt	100%	100%	0%	25%	**100%**	50%	0%	0%	0%	nvt
16	100%	100%	100%	100%	nvt	nvt	100%	100%	100%	100%	100%	100%	100%	100%	100%	nvt
17	100%	100%	100%	100%	nvt	**nvt**	100%	100%	100%	100%	100%	100%	100%	100%	100%	nvt
18	100%	100%	100%	100%	nvt	**nvt**	100%	100%	0%	0%	0%	0%	0%	0%	0%	nvt
19	100%	100%	100%	100%	nvt	nvt	100%	100%	50%	75%	50%	50%	100%	0%	100%	**nvt**
20	100%	100%	100%	100%	**nvt**	**nvt**	100%	100%	100%	100%	100%	75%	100%	0%	**0%**	nvt
21	100%	100%	100%	100%	nvt	nvt	100%	100%	100%	100%	100%	100%	100%	100%	**100%**	**nvt**
22	100%	100%	100%	100%	nvt	nvt	100%	100%	100%	100%	100%	100%	100%	100%	100%	nvt
23	100%	100%	100%	100%	nvt	**nvt**	100%	100%	100%	100%	100%	25%	100%	0%	0%	**nvt**
24	100%	100%	100%	100%	nvt	**nvt**	100%	100%	0%	0%	100%	75%	100%	0%	**0%**	**nvt**
25	100%	100%	100%	100%	nvt	nvt	100%	100%	100%	100%	100%	75%	100%	0%	**100%**	nvt
26	100%	25%	50%	0%	nvt	**nvt**	0%	0%	50%	50%	100%	25%	100%	0%	0%	**nvt**
gemiddeld miscon.	0%	0%	0%	0%	12%	58%	0%	0%	23%	8%	4%	0%	0%	0%	27%	19%
gemiddeld gemaakt	100%	96%	98%	96%	nvt	nvt	96%	92%	62%	66%	85%	54%	69%	35%	42%	nvt

Tabel 7.2: Aanwezige misconcepties getoetst in de toets (horizontaal) per leerling (verticale as). Rood geeft aan dat deze misconceptie gedetecteerd is bij de leerling. Het percentage geeft het percentage gemaakte opdrachten van het lesmateriaal, waarbij de misconceptie behandeld werd, weer. Een *nvt* is ingevuld bij een misconceptie wanneer deze niet in het lesmateriaal opgenomen is of niet gekoppeld is aan enkele opdrachten. De laatste twee rijen geven respectievelijk het percentage leerlingen weer waar misconcepties nog gedetecteerd zijn en het percentage gemaakte opdrachten van het lesmateriaal waar deze misconceptie behandeld werd.

7.3 Vragenlijst

In totaal hebben 30 leerlingen de vragenlijst anoniem ingevuld. Van de leerlingen die de vragenlijst hebben ingevuld, gaven 56.7% aan dat ze al ervaring hadden met visuele programmeertalen en 26.7% al ervaring hadden met een tekstuele programmeertalen voorafgaand deze lessenserie. Daarnaast zijn er vragen gesteld over de lessenserie in het algemeen. Deze resultaten zijn te vinden in Tabel 7.3. Hier valt op dat leerlingen aangeven dat ze de opdrachten leuk (3.73) en relevant (3.73) vonden. Daarnaast geven ze ook aan dat ze het gevoel hebben iets nieuws te hebben geleerd (4.63).

Ook is per hoofdstuk gevraagd wat de leerlingen van het lesmateriaal vonden. Deze resultaten zijn te vinden in Appendix A.5. Hoofdstuk 1 scoort hier het beste. Verder valt op dat latere hoofdstukken slechter beoordeelt worden dan eerdere. Zo zijn leerlingen het meer oneens dat H6 duidelijk is dan eens (2.69). Ook neigde dit hoofdstuk meer naar moeilijk dan het voorgaande hoofdstuk (2.17 tegenover 2.67 bij hoofdstuk 5). Hoofdstuk 7 scoort echter hoger dan het voorgaande hoofdstuk. In Hoofdstuk 8 gaan we verder hier op in.

De vragenlijst vroeg naast vragen gericht op het evalueren van ontwerpeisen, ook de vraag of leerlingen nog overige opmerkingen hadden. 3 van de 30 leerlingen die de vragenlijst hebben ingevuld gaven aan bij deze vraag dat ze graag klassikale uitleg hadden gewild.

Vraag	**N**	**Avg**	**Std**
Ik vond de opdrachten leuk	30	3.73	0.94
Ik vond de opdrachten relevant	30	3.73	0.91
Ik vond de opdrachten duidelijk	30	3.33	0.84
Ik heb iets nieuws geleerd	30	4.63	0.67
Ik heb leren programmeren	30	4.03	0.76
Ik heb het meest geleerd van de uitleg in de opdrachten	30	3.00	0.98
Ik heb het meest geleerd van het maken van de opdrachten	30	4.13	0.73
Ik heb het meest geleerd van de uitleg van de docent	30	2.63	0.72
Ik vond de theorie duidelijk uitgelegd	30	3.60	0.89
Hoe moeilijk vond je de opdrachten in het algemeen?	30	3.47	1.25
Wat vond je van de hoeveelheid opdrachten?	30	3.50	0.82
Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?	30	3.07	1.14

Tabel 7.3: Antwoorden algemene vragen questionnaire. Score gaat van *oneens*, *moeilijk* en *te weinig* (1) t/m *eens*, *makkelijk* en *te veel* (5).

7.3.1 Planning

Oorspronkelijk stonden 12 lessen gepland voor de pilot. Uiteindelijk bleek tijdens het uitvoeren van de pilot maar 11 lessen beschikbaar te zijn. Van deze 11, werd het grootste deel van de laatste les gebruikt voor het maken van de toets en het invullen van de vragenlijst. Effectief bleven dus 10 lessen over. De planning die uiteindelijk gevolgd is, is weergegeven in Tabel 7.4. Omdat H7 geen leerdoelen uit het examenprogramma bevat, is besloten om deze achterwege te laten omwille van de tijd.

	In de les	Na de les
Les 1	intro + start H1	H1 af
Les 2	start H2	H2 af
Les 3	start H3	
Les 4		
Les 5	H3 af	start H4
Les 6		
Les 7		H4 af
Les 8	start H5	
Les 9		H5 af
Les 10	start H6	
Les 11		H6 af

Tabel 7.4: Realisatie lesplanning

Hoofdstuk 8

Discussie

In dit hoofdstuk zullen de resultaten uiteengezet in Hoofdstuk 7 bediscussieerd worden om antwoord te geven in hoeverre voldaan is aan onderzoeksvragen C1 en C2. Daarnaast zullen nog twee algemene bevindingen benoemd worden.

C1: Tot hoever voldoet de mogelijke concrete invulling van het lesmateriaal aan de ontwerpeisen?

Om antwoord te geven op onderzoeksvraag C1, zal per ontwerpeis een discussie volgen in hoeverre aan deze ontwerpeis voldaan is. Alleen ontwerpeisen waaraan niet tijdens de ontwerpfase voldaan is, worden hieronder besproken.

Ontwerpeis 2: Het lesmateriaal moet misconcepties voorkomen en verhelpen. De meeste misconcepties zijn tijdens de toets niet gedetecteerd, echter zijn een aantal nog aanwezig. Misconceptie *V7* blijkt vooral aanwezig te zijn. Een reden voor de hoge aanwezigheid kan zijn dat leerlingen inderdaad deze misconceptie nog hebben. Een andere reden kan de vraag zijn waaruit deze misconceptie gedetecteerd wordt. Bij de vraag krijgen leerlingen een stuk code met de vraag wat er niet aan klopt (een = moet == zijn). Dit wordt in combinatie gedaan met een if-statement en een print. Veel leerlingen maakten de opmerking dat er iets niet met de print klopte (de print was correct). Beargumenteerd kan worden dat deze vraag minder geschikt was om deze misconceptie te detecteren, omdat het lijkt dat leerlingen afgeleid werden van de enkele =. Toch, denken we alsnog dat deze vraag een redelijk beeld geeft of deze misconceptie wel of niet aanwezig is.

Daarnaast was *F6* redelijk aanwezig. Ons eerste vermoeden was dat dit kwam doordat weinig leerlingen de opdrachten hebben gemaakt waar deze misconceptie aan bod kwam. Als gekeken wordt naar Tabel 7.2 valt echter op dat bij de meerderheid, van de leerlingen die 0% gemaakt hebben, deze misconceptie niet gedetecteerd werd. Een andere verklaring waarom *F6* veel aanwezig is, kon niet door ons verzonnen worden.

Als gekeken wordt naar alle misconcepties valt het op dat wanneer een leerling niet de opdrachten heeft gemaakt waarbij een misconceptie aan bod komt, dit niet betekent dat deze aanwezig is bij een leerling. Dit roept de vraag op of de opdrachten, en daarmee het lesmateriaal, daadwerkelijk misconcepties voorkomen/verhelpen. Wegens gebrek aan overtuigend bewijs kunnen we niet stellen dat het lesmateriaal misconcepties voorkomt/verhelpt. Meer onderzoek is nodig op dit vlak. Zo zou een controlegroep inzicht kunnen geven of het aantal aanwezige misconcepties daadwerkelijk gereduceerd is.

Ontwerpeis 3: Het lesmateriaal moet uitdagende en relevante opdrachten met korte termijn doelen bevatten. Tijdens het ontwerpen van het lesmateriaal zijn opdrachten zo gemaakt dat iedere opdracht korte termijn doelen bevat. Uit de vragenlijst komt dat leerlingen gemiddeld genomen het meer eens zijn dat opdrachten relevant zijn dan niet relevant.

Ontwerpeis 7: Het lesmateriaal moet bestaan uit opdrachten die duidelijk en ondubbelzinnig zijn. Uit de vragenlijst komt dat leerlingen gemiddeld genomen meer eens zijn dat de opdrachten duidelijk zijn dan onduidelijk, echter, is dit minimaal (3.33). Daarnaast valt op dat latere hoofdstukken slechter beoordeelt worden dan eerdere. Zo zijn leerlingen het meer oneens dat H6 duidelijk is dan eens. Een verklaring voor het slechter scoren van de latere hoofdstukken kan zijn dat de concepten behandeld in deze hoofdstukken als moeilijker ervaren worden. Lahtinen et al. (2006) benoemt echter

dat de concepten variabelen, keuze en herhaling nagenoeg als even moeilijk worden ervaren door leerlingen. Een andere verklaring kan zijn dat de uitleg minder goed is dan de voorgaande hoofdstukken. Zo scoort H1 bijvoorbeeld het beste van de hoofdstukken. In dit hoofdstuk is de uitleg in de vorm van een video (waar de rest van de hoofdstukken een tekstuele uitleg heeft). Nog een verklaring voor het lager scoren van latere hoofdstukken kan het Learning Edge Momentum zijn: wanneer een concept niet volledig goed aangeleerd wordt, zal een concept die hierop verder bouwt moeilijker zijn om aan te leren.

Op de bovengenoemde observatie is *H7* een uitzondering. We vermoeden dat dit komt doordat leerlingen die aan dit hoofdstuk zijn toegekomen theoretisch sterker zijn. Waren alle leerlingen aan dit hoofdstuk toegekomen, dan had waarschijnlijk dit hoofdstuk lager gescoord.

Ontwerpeis 10: Het lesmateriaal moet geschikt zijn om gegeven te kunnen worden in 12 lessen à 45 minuut per les. Het doel was om het lesmateriaal zo te ontwerpen dat deze in 12 lessen behandeld kan worden. Tijdens de pilot verviel 1 lesuur en werd de laatste les ingenomen door het afnemen van een toets. Daarom hebben we de 12 ingeschatte lessen niet gehaald. Toch denken we dat deze lessenserie in 12 lessen kan. Hierbij moet wel het verdiepende hoofdstuk H7 achterwege gelaten worden. Daarnaast merken we op dat van de latere hoofdstukken er minder opdrachten gemaakt zijn. We vermoeden dat dit deels door tijdgebrek komt. Daarom is het van belang om in het begin het tempo hoog te houden, zodat leerlingen nog genoeg tijd hebben voor de latere hoofdstukken.

Ontwerpeis 11: Het lesmateriaal moet geschikt zijn voor leerlingen van 4 HAVO en 4 VWO. In de vragenlijst die afgenomen is, gaven leerlingen aan dat ze de opdrachten leuk en relevant vonden. Daarnaast geven ze ook aan het gevoel te hebben iets nieuws te hebben geleerd. Wel zien we ruimte voor verbetering. Zo was er net te weinig tijd voor deze lessenserie om alle geplande hoofdstukken te behandelen. Een factor die meegespeeld heeft, is dat veel leerlingen vastliepen op schrijfopdrachten. Hierdoor raakten ze onnodig veel tijd kwijt. Een vermindering in het aantal schrijfopdrachten kan helpen. Beter is, in onze ogen, dat er aandacht besteed wordt aan het oplossen van problemen. Het lesmateriaal besteed op dit moment daar geen aandacht aan. Tijdens de pilot werd vaak de vraag gesteld: *"Meneer, ik weet niet hoe ik verder moet"*. De leerlingen raakten hier duidelijk geërgerd en geïrriteerd door. Aan het lesmateriaal kan een stappenplan toegevoegd worden, wat leerlingen leert om systematisch over een probleem na te denken (Dawkins, Chapman-Brown & Mccuaig, 2019; Hilton, Lipp & Rodger, 2019; Iqbal Malik & Coldwell-Neilson, 2017). Dit stappenplan helpt voorkomen dat leerlingen vastlopen op een probleem en daarmee ook onnodig tijdverlies lijden.

Daarnaast valt op dat met name H6 een stuk minder scoort op duidelijkheid en moeilijkheid. Bij dit hoofdstuk zal de uitleg verbeterd moeten worden en gekeken worden of opdrachten makkelijker gemaakt kunnen worden. Ook zal waarschijnlijk H7 hetzelfde probleem hebben wanneer deze door iedere leerling gemaakt zal worden. Al met al vinden we dat dit lesmateriaal geschikt is voor 4 HAVO en 4 VWO.

Ontwerpeis 12: Het lesmateriaal moet leuk zijn om mee te werken. Uit de vragenlijst blijkt dat leerlingen het gemiddeld genomen meer eens dat de opdrachten leuk zijn dan niet leuk.

C2: Tot hoever beheersen de leerlingen de theorie behandelt in het ontworpen lesmateriaal?

Om antwoord te geven op onderzoeksvraag C2, zal gekeken worden naar de resultaten van de toets.

Uit Tabel 7.1 kunnen we halen dat de eerste vijf vragen goed gemaakt zijn (<25% fout). Vragen 6, 7, 9, 12 en 13 zijn slecht (>50% fout) gemaakt. Een reden waarom vraag 6 slecht gemaakt is, kan komen doordat deze vraag direct een misconceptie test (*V7*: = vs ==). Tijdens het ontwerpen van het lesmateriaal is deze misconceptie over het hoofd gezien en zijn er geen opdrachten in het lesmateriaal voor gekomen. Dit kan een mogelijke reden zijn waarom deze vraag slecht gemaakt is.

Bij vraag 7 wordt een ingewikkelde constructie van geneste if-statements gevraagd. Twee redenen voor de slechte score kunnen hiervoor aangedragen worden. De eerste is dat leerlingen het concept van de if-statement niet tot nauwelijks beheersen (leerdoel C4). Dit is in lijn met het slecht maken van de andere vraag waar dit concept ook getoetst wordt (vraag 6). De tweede reden kan zijn dat leerlingen een fout hebben gemaakt met het uitrekenen van bepaalde waardes. Leerlingen kregen 12 minuten de tijd voor deze toets, dus het zou niet geheel ondenkbaar zijn dat leerlingen in hun haast een foutje hebben gemaakt.

Bij vraag 9 wordt wederom een misconceptie getoetst (wanneer de control variabele niet gebruikt wordt

in een loop, dan wordt deze maar eenmalig uitgevoerd (*H1*)). Hier zijn een verscheidenheid aan verkeerde antwoorden gegeven. Enkele leerlingen gaven als antwoord 4 of 6. Dit betekent waarschijnlijk dat ze de grenzen van de *range()* niet goed in hun hoofd hebben. Een ander deel van de foutieve antwoorden was 1. Dit kan twee dingen betekenen; leerlingen hebben deze misconceptie of leerlingen snappen het concept van een loop nog niet. Als gekeken wordt naar het percentage gemaakte opdrachten (Appendix A.4) valt te zien dat niet alle opdrachten gemaakt zijn van H5. Bij verdere analyse valt op dat leerlingen die vraag 9 goed hadden van H5 gemiddeld meer opdrachten hadden gemaakt dan leerlingen die de vraag fout hadden (19,88 tegenover 15,56 vragen gemaakt van H5).

Vragen 12 en 13 toetsen beide het concept functies (leerdoel C6). Vragen 10 en 11 doen dit ook, maar deze zijn niet zo slecht gemaakt. Een reden voor de grote aantal fouten bij vragen 12 en 13 kan komen, omdat deze vragen een dieper liggende kennis vragen van functies. Kijkend naar het percentage gemaakte opdrachten van H6 valt op dat van dit hoofdstuk minder opdrachten zijn gemaakt, wat gebrek aan kennis kan verklaren.

Gezien het klassen gemiddelde boven de 6 lag en meer dan 60% van de leerlingen een voldoende hadden, stellen we dat de leerdoelen behaald worden. Echter, zijn de aantallen niet groot genoeg om hier een sterk antwoord op te geven. Daarnaast is er ook het feit dat de toets alleen bestond uit leesopdrachten. Een betere toets zou ook aanpas en schrijfopdrachten bevatten. Ook kan het, omdat dat leerlingen geen meetellend cijfer voor de toets kregen, ertoe hebben geleid dat deze minder serieus gemaakt is.

8.1 Algemene Bevindingen

Bij de vragenlijst gaven drie leerlingen aan dat ze graag klassikale uitleg gewild hadden. Dit sluit aan op één van de feedback punten van het expertpanel. Persoonlijk vinden we het interessant om deze opmerking te krijgen van de leerlingen, gezien elke les gevraagd is of leerlingen klassikale uitleg over een concept of opdracht wilden. Dit is nooit benut. Gezien leerlingen er toch behoefte aan hebben is het aan te raden om in het vervolg de concepten toch ook klassikaal uit te leggen. Leerlingen hebben dan twee manieren om een theorie tot zich te nemen: via de uitleg van de docent en via de uitleg in de opdrachten.

Een ander interessant punt is dat leerlingen aangeven het meest te hebben geleerd van het maken van de opdrachten. Deze bevinding sluit aan bij Lahtinen et al. (2006) en het idee achter de lesopbouw.

Hoofdstuk 9

Conclusie

Het doel van dit onderzoek was om een antwoord te geven op de vraag *"Hoe geef je een lessenserie introductie tot programmeren aan 4 HAVO en 4 VWO die misconcepties voorkomt/verhelpt?"*. Aan de hand van de onderzoeksvraag zijn systematisch ontwerpeisen opgesteld, uitgewerkt en gevalideerd. Uiteindelijk concluderen we, aan de hand van wat in dit onderzoek uiteengezet is, dat het beschreven lesmateriaal in de praktijk bruikbaar is voor het geven van programmeren aan 4 HAVO en 4 VWO. Bij dit lesmateriaal leren leerlingen voornamelijk door het maken van de opdrachten. Deze opdrachten gaan in complexiteit van het traceren van een stukje code tot het zelf schrijven van het spelletje Tic-Tac-Toe. Met behulp van de opdrachten behalen leerlingen de leerdoelen en worden misconcepties grotendeels voorkomen/verholpen. Deze conclusie is echter wel gebaseerd op een toets die alleen leesopdrachten bevat. Daarnaast is er geen bewijs of misconcepties voorkomen/verholpen zijn door het lesmateriaal of dat deze überhaupt niet zouden optreden. Leerlingen vonden het over het algemeen leuk om met het materiaal te werken en hebben het gevoel dat ze iets nieuws geleerd hebben. De informatica docent, van de school waar de pilot op gedraaid werd, gaf aan dat deze van plan is het lesmateriaal volgend jaar opnieuw te gebruiken.

Wel zijn er een aantal punten waar het materiaal op verbeterd kan worden. Zo besteed het lesmateriaal geen aandacht aan hoe je vanuit niks een programma schrijft voor een probleemstelling. Leerlingen lopen vaak vast op schrijfopdrachten. Dit kost een leerling veel tijd en frustratie. Een stappenplan dat elke keer doorlopen moet worden om een probleem op te lossen kan hierbij mogelijk helpen. Een ander verbeterpunt is de duidelijkheid van de theorie in de latere hoofdstukken. Leerlingen geven in de vragenlijst aan dat ze de theorie in de latere hoofdstukken minder duidelijk ervaren dan de theorie in de eerdere hoofdstukken.

Uit de toets die de leerlingen moesten maken om te controleren of bepaalde misconcepties nog aanwezig waren, bleek de volgende misconceptie veel aanwezig te zijn: de = als vergelijking operator zien (V7). Tijdens het ontwerpen van het lesmateriaal zijn er geen opdrachten toegevoegd die deze misconceptie behandelen. Daarnaast is de misconceptie: toewijzing van links naar rechts (V6) ook over het hoofd gezien tijdens het ontwerpen. Deze misconceptie was echter in minder grote getalen aanwezig (15 uit 26 voor V7 tegenover 3 uit 26 voor V6). Het toevoegen van opdrachten die deze misconcepties behandelen kan mogelijk helpen in het voorkomen/verhelpen ervan. Echter is het onduidelijk of veel misconcepties voorkomen/verholpen zijn door het lesmateriaal of dat dit ook het geval zou zijn met regulier lesmateriaal. Hier is meer onderzoek voor nodig.

We zijn van mening dat een web-based platform een ideaal hulpmiddel is om lesmateriaal voor programmeren aan te bieden. Wel zien we verbeterpunten voor de toekomst. Allereerst kost het invoeren van de vele opdrachten op Repl.it veel tijd. Ook is het aanpassen van een set opdrachten voor een docent een vervelende taak en ontmoedigt dit om gemaakt werk te verbeteren. We denken dat een Domain Specific Language (DSL) hier veel bij zou kunnen helpen. Op deze manier kunnen opdrachten aangepast worden, zonder het trage invoer/aanpas proces op de website meerdere malen te moeten doorlopen.

Ook biedt, in theorie, een web-based oplossing veel mogelijkheden tot data analyse. Deze analyse zou vervolgens gebruikt worden om het lesmateriaal te verbeteren. Op dit moment slaat Repl.it alleen het eind antwoord van een leerling op, tussenproducten gaan verloren. Hiermee gaat data die inzicht kan geven in het proces van een leerling verloren. Ook zou er gekeken kunnen worden naar hoelang leerlingen gemiddeld genomen met bepaalde opdrachten bezig zijn. Dit kan moeilijke/onduidelijke opdrachten aan het licht brengen.

Bibliografie

Armoni, M., Meerbaum-Salant, O. & Ben-Ari, M. (2015). From Scratch to “Real” Programming. *ACM Transactions on Computing Education*, *14*(4), 1–15. doi: 10.1145/2677087

Ben-Ari, M. (1998). Constructivism in computer science education. *ACM SIGCSE Bulletin*, *30*(1), 257–261. doi: 10.1145/274790.274308

Bennedsen, J. & Caspersen, M. E. (2007). Failure Rates in Introductory Programming. , *39*(2), 32–36.

Bennedsen, J. & Caspersen, M. E. (2019). Failure Rates in Introductory Programming — 12 Years Later. , *10*(2), 30–35.

Bornat, R., Dehnadi, S. & Simon. (2008). Mental models, consistency and programming aptitude. *Proceedings of the Tenth Australasian Computing Education Conference*, *78*, 53–61.

Corney, M. (2009). Designing for engagement: Building IT systems. *ALTC First Year Experience Curriculum Design Symposium 2009*, 19–21.

Dawkins, H., Chapman-Brown, G. & Mccuaig, J. (2019). Nifty Assignment: Problem Solving Workshop. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*(5), 5.

Edwards, J. M., Fulton, E. K., Holmes, J. D., Valentin, J. L., Beard, D. V. & Parker, K. R. (2019). Separation of syntax and problem solving in Introductory Computer Programming. *2018 IEEE Frontiers in Education Conference (FIE)*, 1–5. doi: 10.1109/fie.2018.8658852

Grandell, L., Peltomäki, M., Back, R. J. & Salakoski, T. (2006). Why complicate things? Introducing programming in high school using Python. *Conferences in Research and Practice in Information Technology Series*, *52*, 71–80.

Hassinen, M. & Mäyrä, H. (2006). Learning programming by programming: a case study. *Proceedings of the 6th Baltic Sea conference on . . .*, 117–119. Verkregen van http://dl.acm.org/citation.cfm?id=1315824

Hermans, F., Swidan, A. & Smit, M. (2018). Thinking out of the Box Comparing metaphors for variables in programming education. *Proceedings of the 13th Workshop in Primary and Secondary Computing Education*.

Hilton, A. D., Lipp, G. M. & Rodger, S. H. (2019). Translation from Problem to Code in Seven Steps. *Proceedings of the ACM Conference on Global Computing Education*, 78–84. doi: 10.1145/3300115.3309508

Horstmann, C. (2013). *Big: Late Objects*. John Wiley & Sons Inc.

Hristova, M., Misra, A., Rutter, M. & Mercuri, R. (2003). Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. , 153–156.

Hudak, M. & Anderson, D. (1990). Formal operations and learning style predict success in statistics and computer science courses. *Teaching of Psychology*, *17*(4), 231–234.

Iqbal Malik, S. & Coldwell-Neilson, J. (2017). Comparison of Traditional and ADRI Based Teaching Approaches in an Introductory Programming Course. *Journal of Information Technology Education: Research*, *16*, 267–283. doi: 10.28945/3793

Kallia, M. & Sentance, S. (2019). Learning to use Functions: The Relationship Between Misconceptions and Self-Efficacy. *In Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 752–758. doi: 10.1145/3287324.3287377

Lahtinen, E., Ala-Mutka, K. & Järvinen, H.-M. (2006). A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, *37*(3), 14. doi: 10.1145/1151954.1067453

Ma, L., Ferguson, J., Roper, M. & Wood, M. (2007). Investigating the viability of mental models held by novice programmers. *ACM SIGCSE Bulletin*, *39*(1), 499. doi: 10.1145/1227504.1227481

Meerbaum-Salant, O., Armoni, M. & Ben-Ari, M. (2013). Learning computer science with scratch. *Computer Science Education*, *23*(3), 239–264.

Mohamad Gobil, A. R., Shukor, Z. & Mohtar, I. A. (2009). Novice difficulties in selection structure. *Proceedings of the 2009 International Conference on Electrical Engineering and Informatics, ICEEI 2009*, *2*(August), 351–356. doi: 10.1109/ICEEI.2009.5254715

Parnas, D. L., Shore, J. E. & Weiss, D. (1976). Abstract types defined as classes of variables. *ACM SIGMOD Record*, *8*(2), 149–154.

Pea, R. D., Soloway, E. & C., S. J. (1987). The Buggy Path to The Development of Programming Expertise. , *9*(1), 5–30.

Peters, R. M. (2018). *Identifying and Adressing Common Programming Misconceptions with Variables - Part II* (Master Thesis). Univiersity of Twente, Enschede.

Plass, D. (2015). *Identifying and Addressing Common Programming Misconceptions with Variables (Part 1)* (Academisch proefschrift). Universiteit Twente.

Rajaravivarma, R. (2006). A games-based approach for teaching the introductory programming course. *ACM SIGCSE Bulletin*, *37*(4), 98. doi: 10.1145/1113847.1113886

Robins, A. (2010). *Learning edge momentum: A new account of outcomes in CS1* (Dl. 20) (nr. 1). doi: 10.1080/08993401003612167

Sekiya, T. & Yamaguchi, K. (2013). Tracing quiz set to identify novices' programming misconceptions. , 87–95. doi: 10.1145/2526968.2526978

SLO. (2018a). *Domein d: Programmeren.* http://handreikingschoolexamen.slo.nl/informatica/Paginas/Domein-D.aspx. (Accessed: 2019-03-24)

SLO. (2018b). *Voorbeeldspecificaties domein d.* http://handreikingschoolexamen.slo.nl/informatica/Paginas/Voorbeeldspecificaties-domein-D.aspx. (Accessed: 2019-03-24)

Sorva, J. (2009). The same but different students' understandings of primitive and object variables. , 5. doi: 10.1145/1595356.1595360

Spohrer, J. C. & Soloway, E. (1986). Novice mistakes: are the folk wisdoms correct? *Communications of the ACM*, *29*(7), 624–632. doi: 10.1145/6138.6145

Staubitz, T., Klement, H., Renz, J., Teusner, R. & Meinel, C. (2016). Towards practical programming exercises and automated assessment in Massive Open Online Courses. *Proceedings of 2015 IEEE International Conference on Teaching, Assessment and Learning for Engineering, TALE 2015*(December), 23–30. doi: 10.1109/TALE.2015.7386010

Tang, T., Rixner, S. & Warren, J. (2014). An environment for learning interactive programming. , 671–676. doi: 10.1145/2538862.2538908

Triglianos, V., Labaj, M., Moro, R., Simko, J., Hucko, M., Tvarozek, J., . . . Bielikova, M. (2017). Experiences Using an Interactive Presentation Platform in a Functional and Logic Programming Course. *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*, 311–316. doi: 10.1145/3099023.3099082

Vihavainen, A., Paksula, M. & Luukkainen, M. (2011). Extreme Apprenticeship Method in Teaching Programming for Beginners. *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, 93–98.

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., . . . Ko, A. J. (2019). A theory of instruction for introductory programming skills. *Computer Science Education*, *00*(00), 1–49. Verkregen van https://doi.org/10.1080/08993408.2019.1565235 doi: 10.1080/08993408.2019.1565235

Appendices

A.1 Misconcepties Overzicht

V1 Oplossen wiskundige formule

V2 Variabele kan maar 1 maal toegewezen worden

V3 Variabele naam heeft invloed op waarde van variabele

V4 Variabele toewijzen aan andere variabele zorgt ervoor dat eerste variabele zijn waarde kwijtraakt.

V5 Variabele kan meerdere waardes tegelijk bevatten

V6 Toewijzing van links naar rechts

V7 De = als vergelijking operator zien

V8 Bij een toewijzing de waardes van twee variabelen verwisseld worden

V9 Resultaten kunnen alleen opgeslagen worden in een variabele niet genoemd in de expressie

H1 Wanneer in een loop de control variabele niet gebruikt wordt, dan wordt de loop maar 1 keer uitgevoerd

H2 Voorgaande waardes van een variabele in een loop blijven bewaard

H3 Wanneer een control variabele false wordt binnen de body van de loop, stopt de loop meteen met uitvoeren.

F1 Return statement moet gebruikt worden na de aanroep van de functie om de waarde van de functie te printen en terug te geven.

F2 Return variabele moet dezelfde naam hebben als de naam van de functie

F3 Parameters moeten expliciet gedefinieerd worden om gebruikt te kunnen worden binnen een functie

F4 Parameters en argumenten moeten dezelfde naam hebben

F5 Argumenten moeten ook de return waarde bevatten

F6 Locale variabelen kunnen van functie tot functie benaderd worden

F7 Het aanroepen van een functie geeft het resultaat weer

A.2 Oefentoets

Introductie Programmeren oefentoets

Naam: _____

28 mei 2019

1. *(5 punten)* Wat is de uiteindelijke waarde van variabel *a* na het uitvoeren van onderstaande programma?

```
b = 10
a = b / 2
```

```
Antwoord:
```

2. *(5 punten)* Wat is de uiteindelijke waarde van variabel *a* na het uitvoeren van onderstaande programma?

```
a = 10
a = 20
```

```
Antwoord:
```

3. *(5 punten)* Wat is de uiteindelijke waarde van variabel *tien* na het uitvoeren van onderstaande programma?

```
tien = 20
```

```
Antwoord:
```

4. *(5 punten)* Wat is de uiteindelijke waarde van variabel *a* na het uitvoeren van onderstaande programma?

```
a = 15
b = 5
b = a
```

```
Antwoord:
```

5. *(10 punten)* Wat is de uiteindelijke waarde van variabel *a* na het uitvoeren van onderstaande programma wanneer de gebruiker eerst *5* invult en vervolgens *10*?

```
waarde1 = float(input())
waarde2 = float(input())
a = waarde1 + waarde2
```

```
Antwoord:
```

6. *(10 punten)* Onderstaande programma moet *a is 0* weergeven. Geef aan wat fout is aan onderstaande programma (*(5 punten)* en waarom dit fout is *(10 punten)*.

```
a = 0
if a = 0:
  print("a is 0")
```

```
Antwoord:
```

7. *(10 punten)* Wat is de uiteindelijke waarde van variabel *a* na het uitvoeren van onderstaande programma?

```
a = 30
if a/10==3:
  if a*3<100:
    if a+20<40:
      a = 0
    else:
      a = 100
  else:
    a = 150
else:
  a = 64
```

```
Antwoord:
```

8. *(10 punten)* Wat is de uiteindelijke waarde van variabel a na het uitvoeren van onderstaande programma?

```
a = 0
b = 0
while b != 4:
  b = 4
  a = a+1
```

```
Antwoord:
```

9. *(10 punten)* Wat is de uiteindelijke waarde van variabel a na het uitvoeren van onderstaande programma?

```
a = 0
for i in range(0,5):
  a = a + 1
```

```
Antwoord:
```

10. *(10 punten)* Wat geeft onderstaande programma weer.

```
def hoi():
  print("hallo")

hoi()
```

```
Antwoord:
```

11. *(10 punten)* Wat geeft onderstaande programma weer.

```
def random():
  return 4

print(random())
```

```
Antwoord:
```

12. *(10 punten)* Wat geeft onderstaande programma weer.

```
def p(waarde):
  print("hallo")
  return 10
  print("wereld")

p(5)
```

```
Antwoord:
```

13. *(10 punten)* Wat is de uiteindelijke waarde van variabel a na het uitvoeren van onderstaande programma?

```
def f1():
  a = 32
  return 18

def f2(waarde):
  a = 16
  return f1(a)

a = f2(42)
```

```
Antwoord:
```

A.3 Vragenlijst

Programmeer Ervaring

* Required

1. **Ik had vooraf deze lessen ervaring visuele programmeertalen** *
   Een visuele programmeertaal is een taal waarbij met blokken geprogrammeerd wordt. Dit hebben jullie de eerste les gedaan met Flappy Bird of het Minecraft poppetje. Deze soort taal wordt bijvoorbeeld ook gebruikt bij LEGO Mindstorms.
   Mark only one oval.

   - Ja
   - Nee

2. **Ik had vooraf deze lessen ervaring tekstuele programmeertalen** *
   Een tekstuele programmeertaal is een taal waarbij met tekst geprogrammeerd wordt (zoals bij Python, Java, C++, C#).
   Mark only one oval.

   - Ja
   - Nee

Gemaakte Lesmateriaal

Geef aan hoe ver je eens bent met onderstaande stellingen

3. **Ik vond de opdrachten leuk** *
   Mark only one oval.

	1	2	3	4	5	
oneens						eens

4. **Ik vond de opdrachten relevant** *
   Mark only one oval.

	1	2	3	4	5	
oneens						eens

5. **Ik vond de opdrachten duidelijk** *
   Mark only one oval.

	1	2	3	4	5	
oneens						eens

6. **Ik heb iets nieuws geleerd** *
   Mark only one oval.

	1	2	3	4	5	
oneens						eens

7. **Ik heb leren programmeren** *
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

8. **Ik heb het meest geleerd van de uitleg in de opdrachten** *
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

9. **Ik heb het meest geleerd van het maken van de opdrachten** *
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

10. **Ik heb het meest geleerd van de uitleg van de docent** *
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

Evaluatie per hoofdstuk

BELANGRIJK: Op het moment dat je niet aan een hoofdstuk bent toegekomen, hoef je van dat hoofdstuk niks in te vullen

Hoofdstuk 0: Printen

11. **Ik vond de theorie duidelijk uitgelegd**
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

12. **Hoe moeilijk vond je de opdrachten in het algemeen?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

13. **Wat vond je van de hoeveelheid opdrachten?**
Mark only one oval.

	1	2	3	4	5	
te weinig	◯	◯	◯	◯	◯	te veel

14. **Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

Hoofdstuk 1: Variabelen

15. **Ik vond de theorie duidelijk uitgelegd**
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

16. **Hoe moeilijk vond je de opdrachten in het algemeen?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

17. **Wat vond je van de hoeveelheid opdrachten?**
Mark only one oval.

	1	2	3	4	5	
te weinig	◯	◯	◯	◯	◯	te veel

18. **Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

Hoofdstuk 2: Types & Operaties

19. **Ik vond de theorie duidelijk uitgelegd**
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

20. **Hoe moeilijk vond je de opdrachten in het algemeen?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

21. **Wat vond je van de hoeveelheid opdrachten?**
Mark only one oval.

	1	2	3	4	5	
te weinig	◯	◯	◯	◯	◯	te veel

22. **Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

Hoofdstuk 3: Input & Casting

23. **Ik vond de theorie duidelijk uitgelegd**
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

24. **Hoe moeilijk vond je de opdrachten in het algemeen?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

25. **Wat vond je van de hoeveelheid opdrachten?**
Mark only one oval.

	1	2	3	4	5	
te weinig	◯	◯	◯	◯	◯	te veel

26. **Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

Hoofdstuk 4: Keuze

27. **Ik vond de theorie duidelijk uitgelegd**
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

28. **Hoe moeilijk vond je de opdrachten in het algemeen?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

29. **Wat vond je van de hoeveelheid opdrachten?**
Mark only one oval.

	1	2	3	4	5	
te weinig	◯	◯	◯	◯	◯	te veel

30. **Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

Hoofdstuk 5: Herhaling

31. **Ik vond de theorie duidelijk uitgelegd**
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

32. **Hoe moeilijk vond je de opdrachten in het algemeen?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

33. **Wat vond je van de hoeveelheid opdrachten?**
Mark only one oval.

	1	2	3	4	5	
te weinig	◯	◯	◯	◯	◯	te veel

34. **Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

Hoofdstuk 6: Functies

35. **Ik vond de theorie duidelijk uitgelegd**
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

36. **Hoe moeilijk vond je de opdrachten in het algemeen?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

37. **Wat vond je van de hoeveelheid opdrachten?**
Mark only one oval.

	1	2	3	4	5	
te weinig	◯	◯	◯	◯	◯	te veel

38. **Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

Hoofdstuk 7: Lijsten

39. **Ik vond de theorie duidelijk uitgelegd**
Mark only one oval.

	1	2	3	4	5	
oneens	◯	◯	◯	◯	◯	eens

40. **Hoe moeilijk vond je de opdrachten in het algemeen?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

41. **Wat vond je van de hoeveelheid opdrachten?**
Mark only one oval.

	1	2	3	4	5	
te weinig	◯	◯	◯	◯	◯	te veel

42. **Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

Hoofdstuk 8: Extra opdrachten

43. **Hoe moeilijk vond je de opdrachten?**
Mark only one oval.

	1	2	3	4	5	
moeilijk	◯	◯	◯	◯	◯	makkelijk

Overig

Mocht je een opmerkingen hebben over waarom een specifiek hoofdstuk makkelijk/moeilijk was, kan je deze hieronder kwijt.

44. **Opmerkingen over je antwoord**

_____

_____

_____

_____

_____

Overig

Mocht je nog dingen hebben die je graag kwijt wil, kan je deze hier noemen.

45. **Overige opmerkingen**

Powered by
Google Forms

A.4 Percentage gemaakte opdrachten per hoofdstuk

	H0			H1			H2			H3			H4			H5		
lln	l	a	s	l	a	s	l	a	s	l	a	s	l	a	s	l	a	s
1	100%	100%	100%	100%	100%	100%	100%	33%	100%	57%	14%	75%	15%	0%	40%	23%	0%	22%
2	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	86%	100%	100%	100%	100%	100%	100%	67%
3	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	86%	60%	15%	50%	56%
4	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	44%
5	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	38%	0%	33%
6	100%	100%	100%	100%	100%	100%	42%	67%	100%	71%	86%	50%	38%	0%	80%	23%	0%	44%
7	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	75%	100%	100%	100%	85%	0%	0%
8	100%	0%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	46%	33%	78%
9	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	78%
10	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	40%	23%	0%	11%
11	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	77%	17%	22%
12	100%	100%	100%	100%	100%	100%	83%	100%	100%	100%	100%	100%	100%	71%	100%	92%	83%	78%
13	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	78%
14	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
15	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	38%	0%	56%
16	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	78%
17	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	60%	100%	100%	33%
18	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	0%	8%	0%	0%
19	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	77%	17%	22%
20	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	83%	67%
21	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	44%
22	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	78%
23	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	44%
24	100%	100%	100%	100%	86%	100%	67%	100%	100%	100%	100%	100%	54%	14%	80%	23%	83%	33%
25	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	77%	0%	89%
26	100%	100%	100%	57%	57%	100%	67%	67%	100%	100%	86%	100%	54%	71%	100%	77%	83%	33%

Tabel 1: Percentage gemaakte opdrachten van H0 t/m H5 per type opdracht: (l)ezen, (a)anpassen en (s)chrijven.

	H6			H7			H8		
lln	l	a	s	l	a	s	l	a	s
1	47%	0%	40%	0%	0%	0%	0%	0%	0%
2	13%	0%	60%	0%	0%	0%	0%	0%	0%
3	73%	100%	40%	10%	0%	0%	0%	0%	0%
4	100%	100%	40%	30%	0%	0%	0%	0%	0%
5	40%	33%	40%	0%	0%	0%	0%	0%	0%
6	67%	0%	40%	0%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	40%	50%	40%	0%	0%	0%	0%	0%	0%
9	0%	0%	40%	0%	0%	0%	0%	0%	0%
10	20%	0%	0%	60%	0%	0%	11%	0%	0%
11	20%	0%	40%	0%	0%	0%	0%	0%	0%
12	73%	67%	40%	70%	50%	14%	0%	0%	0%
13	100%	100%	40%	0%	0%	0%	0%	0%	0%
14	100%	100%	100%	100%	100%	71%	33%	0%	0%
15	40%	33%	40%	0%	0%	0%	0%	0%	0%
16	100%	100%	40%	70%	25%	14%	0%	0%	0%
17	100%	67%	40%	0%	0%	0%	0%	0%	0%
18	0%	0%	0%	0%	0%	0%	0%	0%	0%
19	33%	0%	40%	0%	0%	0%	0%	0%	0%
20	73%	0%	40%	40%	0%	29%	0%	0%	0%
21	100%	100%	40%	0%	0%	0%	0%	0%	0%
22	100%	100%	40%	100%	0%	0%	0%	0%	0%
23	33%	0%	40%	0%	0%	0%	0%	0%	0%
24	80%	0%	40%	30%	0%	0%	11%	0%	0%
25	93%	0%	100%	70%	0%	57%	22%	0%	0%
26	40%	17%	40%	90%	75%	0%	0%	0%	0%

Tabel 2: Percentage gemaakte opdrachten van H6 t/m H8 per type opdracht: (l)ezen, (a)anpassen en (s)chrijven.

A.5 Resultaat vragenlijst per hoofdstuk

Vraag	N	Avg	Std
Hoofdstuk 0			
Ik vond de theorie duidelijk uitgelegd	30	3.60	0.89
Hoe moeilijk vond je de opdrachten in het algemeen?	30	3.47	1.25
Wat vond je van de hoeveelheid opdrachten?	30	3.50	0.82
Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?	30	3.07	1.14
Hoofdstuk 1			
Ik vond de theorie duidelijk uitgelegd	30	3.80	0.89
Hoe moeilijk vond je de opdrachten in het algemeen?	30	3.40	1.16
Wat vond je van de hoeveelheid opdrachten?	30	3.20	0.66
Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?	30	3.43	1.04
Hoofdstuk 2			
Ik vond de theorie duidelijk uitgelegd	30	3.37	0.96
Hoe moeilijk vond je de opdrachten in het algemeen?	30	3.17	0.95
Wat vond je van de hoeveelheid opdrachten?	30	3.30	0.65
Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?	30	3.20	0.96
Hoofdstuk 3			
Ik vond de theorie duidelijk uitgelegd	30	3.33	0.99
Hoe moeilijk vond je de opdrachten in het algemeen?	30	3.00	0.83
Wat vond je van de hoeveelheid opdrachten?	30	3.40	0.56
Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?	30	3.03	0.89
Hoofdstuk 4			
Ik vond de theorie duidelijk uitgelegd	30	3.27	1.01
Hoe moeilijk vond je de opdrachten in het algemeen?	30	2.80	0.89
Wat vond je van de hoeveelheid opdrachten?	30	3.47	0.57
Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?	30	2.80	1.00
Hoofdstuk 5			
Ik vond de theorie duidelijk uitgelegd	30	3.10	1.03
Hoe moeilijk vond je de opdrachten in het algemeen?	30	2.67	0.96
Wat vond je van de hoeveelheid opdrachten?	30	3.47	0.78
Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?	30	2.53	0.97
Hoofdstuk 6			
Ik vond de theorie duidelijk uitgelegd	29	2.69	1.11
Hoe moeilijk vond je de opdrachten in het algemeen?	29	2.17	0.93
Wat vond je van de hoeveelheid opdrachten?	29	3.48	0.69
Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?	28	2.18	0.90
Hoofdstuk 7			
Ik vond de theorie duidelijk uitgelegd	13	3.31	1.18
Hoe moeilijk vond je de opdrachten in het algemeen?	13	2.69	1.18
Wat vond je van de hoeveelheid opdrachten?	11	3.18	1.08
Hoe moeilijk vond je het toepassen van de theorie (het volledig zelf schrijven)?	11	2.45	0.82
Hoofdstuk 8			
Hoe moeilijk vond je de opdrachten?	10	2.30	0.95

Tabel 3: Antwoorden per hoofdstuk. Score gaat van *oneens*, *moeilijk* en *te weinig* (1) t/m *eens*, *makkelijk* en *te veel* (5).

A.6 Opdrachten

In dit hoofdstuk staan de opdrachten beschreven zoals ze in Repl.it zijn ingevuld. De titels van de opdrachten volgen een vaste structuur van drie getallen. De eerste geeft het hoofdstuknummer aan, de tweede welk type opdracht (traceren, aanpassen en schrijven) en het derde getal geeft het opdrachtnummer aan. Verder hebben sommige titels een *, wat betekent dat ze verplicht zijn voor de leerlingen, en een +, wat betekent dat deze opdracht meetelt voor hun cijfer.

Verder zit er in de opdrachten ook een vaste structuur. Het deel tot het kopje *code* wordt weergegeven op het uitlegscherm van de opdracht. De code onder *code* wordt al ingevuld voor de leerlingen in het Code Editor. Omdat er verschillende soorten testen zijn die Repl.it biedt (input- en output-matching en unit testing), is er voor gekozen om deze niet in het verslag op te nemen.

A.6.1 Printing

[0.0.0] Introductie*

Programmeren In deze lessen gaan jullie leren programmeren. Bij programmeren maak je een programma. Een programma is een stappenplan wat je computer opvolgt. Dit stappenplan moet in een taal (die begrijpbaar is voor een computer) geschreven worden. In deze lessen gaan jullie de taal Python leren.

Code Editor Een *Code Editor* is iets waar je je programma in opschrijft (vergelijkbaar met Word). Op deze pagina is je Code Editor het venster linksboven. In dat venster schrijf je je programma. Dat kun je vervolgens uitvoeren door op de knop “run” te klikken boven dit venster. Het resultaat van je programma wordt weergeven in het *Uitvoerscherm*.

Uitvoerscherm Het *Uitvoerscherm* is waar het resultaat van je programma wordt weergeven. Het resultaat van je programma zal hier verschijnen nadat je op de knop “run” hebt gedrukt. Op deze pagina is het Uitvoerscherm het venster linksonder. Standaard staat hier onderstaande tekst weergeven, dit mag je negeren.

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
```

Opdrachtnummering Alle opdrachten hebben een nummering gebruik makend van 3 getallen. De eerste geeft het hoofdstuknummer aan. Het tweede getal geeft het soort opdracht aan en het derde getal geeft het opdrachtnummer aan.

Daarnaast kan er nog een * of een + achter de titel staan. Een * geeft aan dat deze verplicht is om te maken. Een + geeft aan dat deze opdracht voor je cijfer meetelt. Je bent dus niet verplicht alle opdrachten te maken. Denk je dat je genoeg geoefend hebt met een bepaalde soort opdracht, dan kan je de rest van de opdrachten van dat soort overslaan. Kom je er achter dat je toch nog wel moeite hebt met een onderwerp, kan je altijd nog terug gaan.

Hoofdstukken Hieronder staat weergegeven in welk hoofdstuknummer welk onderwerp aan bod komt.

0 Printen

1 Variabelen

2 Types & operaties

3 Input & casting

4 Keuze

5 Herhaling

6 Functies

7 Lijsten

8 Extra opdrachten

Opdracht
Druk op *submit* om de opdracht in te leveren.

Code

```

```

[0.0.1] Je eerste programma*

Als eerste programma ga je een programma maken die de tekst *hallo wereld* weergeeft. De code die dit uitvoert is al voor je ingevuld in de *Code Editor*.

Opdracht
Druk op run. Observeer dat dit programma inderdaad *hallo wereld* weergeeft op het Uitvoerscherm.

Hoe weet de computer dat deze hallo wereld weer moet geven op het Uitvoerscherm? De computer begrijpt dat wanneer je *print()* typt, alles wat tussen haakjes staat moet worden weergegeven op het Uitvoerscherm

Waarom geeft de computer niet de "weer? Dit heeft te maken met *variabelen* waar jullie in het volgend hoofdstuk over zullen leren. Voor nu moet je onthouden dat alles wat je weer wilt geven tussen "moet staan.

Code

```
1 print("hallo wereld")
```

[0.0.2] Aanhalingstekens

Opdracht
Wat is de uitvoer van onderstaande programma? Typ de code over in de Code Editor.

```
1 print("\"hallo wereld\"")
```

Code

```
1
```

[0.1.0] Error

Gegeven is het programma dat probeert *niks* weer te geven op het Uitvoerscherm.

```
1 hallo Bob
```

Opdracht
Voer het programma uit. Onderstaande zal worden weergegeven op het Uitvoerscherm. Wat dit bericht precies inhoudt zal je later leren.

```
1 Traceback (most recent call last): File "python", line 1, in <module> NameError: name 'niks'
    is not defined
```

Pas het programma aan zodat het *niks* weergeeft op het Uitvoerscherm.

Code

```
1 print(niks)
```

[0.1.1] Hallo

Opdracht
Pas de gegeven programma aan zodat deze hallo <je eigen naam> weergeeft.

Voorbeeld: Stel je naam is Bob. Dan geeft dit programma het volgende weer op het Uitvoer Scherm.

```
hallo Bob
```

Code

```
print("hallo wereld")
```

[0.1.2] "hallo Bob"

Opdracht
Pas het gegeven programma aan zodat het onderstaande weergeeft.

```
"hallo Bob"
```

Code

```
print("hallo Bob")
```

[0.2.0] Hallo Alice+

Opdracht
Schrijf een programma dat *hallo alice* weergeeft.

Code

```

```

[0.2.1] "Hoi Kitty"+

Opdracht
Schrijf een programma dat *"hoi kitty"* weergeeft.

Code

```

```

A.6.2 Variabelen

[1.0.0] Variabelen*

Kijk de YouTube video: https://www.youtube.com/watch?v=WARZCZ_D66Y

In dit filmpje wordt onderstaande stuk code gebruikt om geheugen te reserveren. In veel talen is het nodig om geheugen te reserveren voor een variabel. In Python hoeft dit niet, Python doet dit automatisch voor jou.

```
Dim lengte AS Integer
```

Opdracht
Wat is de uiteindelijke waarde van de variabel jam?

```
jam = "a"
pindakaas = "b"
jam = "c"
```

Hoe antwoorden: Stel je denkt dat jam als waarde *antwoord* heeft, vul dan onderstaande code in de Code Editor.

```
print("de waarde van jam is antwoord")
```

Code

```
print("de waarde van jam is ...")
```

[1.0.1] hallo wereld 2

Opdracht
Wat geeft onderstaande programma weer op het Uitvoerscherm?

```
tekst = "hallo wereld"
print(tekst)
```

Hoe antwoorden: Stel je denkt dat dit programma *hoi bob* weergeeft, vul dan onderstaande code in de Code Editor.

```
print("dit programma geeft hoi bob weer")
```

Code

```
print("dit programma geeft ... weer")
```

[1.0.2] tekst

Opdracht
Wat geeft onderstaande programma weer op het Uitvoerscherm?

```
tekst = "hallo wereld"
print("tekst")
```

Hoe antwoorden: Stel je denkt dat dit programma *hoi bob* weergeeft, vul dan onderstaande code in de Code Editor.

```
print("dit programma geeft hoi bob weer")
```

Code

```
print("dit programma geeft ... weer")
```

[1.0.3] twee

Opdracht
Wat is de uiteindelijke waarde van de variabel twee?

```
twee = "drie"
```

Hoe antwoorden: Stel je denkt dat jam als waarde *antwoord* heeft, vul dan onderstaande code in de Code Editor.

```
print("de waarde van jam is antwoord")
```

Code

```
print("de waarde van twee is ...")
```

[1.0.4] Switcheroo

Opdracht
Wat zijn de uiteindelijke waarden van de variabel a en b?

```
a = "tekst"
b = "woord"
a = b
b = a
```

Hoe antwoorden: Stel je denkt dat a als waarde *antwoord* heeft en b als waarde *iets*, vul dan onderstaande code in de Code Editor.

```
print("de waarde van a is antwoord")
print("de waarde van b is iets")
```

Code

```
print("de waarde van a is ...")
print("de waarde van b is ...")
```

[1.0.5] Switcheroo 2

Opdracht
Wat zijn de uiteindelijke waarden van de variabel a en b?

```
a = "tekst"
b = "woord"
b = a
a = b
```

Hoe antwoorden: Stel je denkt dat a als waarde *antwoord* heeft en b als waarde *iets*, vul dan onderstaande code in de Code Editor.

```
print("de waarde van a is antwoord")
print("de waarde van b is iets")
```

Code

```
print("de waarde van a is ...")
print("de waarde van b is ...")
```

[1.0.6] Maximum en Minimum

Opdracht
Wat zijn de uiteindelijke waarden van de variabel maximum en minimum?

```
maximum = "minimum"
minimum = "maximum"
```

Hoe antwoorden: Stel je denkt dat a als waarde *antwoord* heeft en b als waarde *iets*, vul dan onderstaande code in de Code Editor.

```
print("de waarde van maximum is antwoord")
print("de waarde van minimum is iets")
```

Code

```
print("de waarde van maximum is ...")
print("de waarde van minimum is ...")
```

[1.1.0] Hallo Alice 2

Opdracht
Pas het programma aan zodat het *hallo alice* weergeeft. Doe dit zonder de laatste regel van het programma aan te passen.

Code

```
variable = "hallo bob"
print(variable)
```

[1.1.1] Regel te veel

Opdracht
Pas het programma aan zodat het *a* weergeeft. Doe door 1 regel te verwijderen.

Code

```
a = "a"
b = "b"
a = b
print(a)
```

[1.1.2] Wisseltruc

Opdracht
Pas het programma aan zodat het *a* weergeeft. Doe dit door twee regels om te draaien van plek. Laat de *print(a)* onveranderd van plek.

Code

```
a = "a"
b = "b"
c = "c"
a = b
b = a
print(a)
```

[1.1.3] Eén regel fout

Opdracht
Pas het programma aan zodat het *c* weergeeft. Doe dit door alleen regel 4 te veranderen.

Code

```
a = "a"
b = "b"
c = "c"
a = b
print(a)
```

[1.1.4] Wisseltruc 2

Opdracht
Pas het programma aan zodat het *a* weergeeft. Doe dit door twee regels om te draaien van plek. Laat de *print(a)* onveranderd van plek.

Code

```
1 a = "a"
  b = "b"
3 c = "c"
  a = b
5 b = a
  print(a)
```

[1.1.5] Eén regel fout 2

Opdracht
Pas het programma aan zodat het *c* weergeeft. Doe dit door alleen regel 2 te veranderen.

Code

```
  a = "a"
2 b = "b"
  c = "c"
4 a = b
  print(a)
```

[1.1.6] Wisseltruc 3

Opdracht
Pas het programma aan zodat het *b* weergeeft. Doe dit door twee regels om te draaien van plek.

Code

```
1 a = "a"
  b = "b"
3 print(a)
  a = b
```

[1.2.0] Mijn Hallo+

Opdracht
Schrijf een programma die de waarde *hallo bob* in een variabele stopt. Geef vervolgens de waarde van deze variabel weer.

Code

```

```

[1.2.1] Mijn Variabelen+

Opdracht
Schrijf een programma waar je de variabel b de waarde *a* geeft. Geef vervolgens de waarde van b weer.

Code

```
1
```

A.6.3 Types en expressies

[2.0.0] Types en expressies*

Type Tot nu toe hebben we nog maar met 1 soort (in de informatica ook wel *type* genoemd) variabel gewerkt. Dit is het type *string*. Een string (niet te verwarren met het ondergoed) bevat tekst. Hieronder staan nog twee veelgebruikte variabeltypen uitgelegd.

Boolean Een boolean is een variabel die maar 2 mogelijkheden kan hebben; *waar* en onwaar. Hieronder is weergegeven hoe dit in Python eruitziet.

```
waar = True
onwaar = False
```

Bij strings moesten we altijd ” om de waarde heen zetten. Zo weet de computer dat je een string bedoeld. Doe je dit niet, dan denkt de computer dat je een variabel bedoeld. Bij booleans heb je maar twee waarden. Daarom heeft Python twee woorden (*True* voor waar en *False* voor onwaar) gereserveerd voor de waardes van booleans. Dit betekent dus ook dat je geen variabelnaam *True* of *False* kan hebben. Let hierbij op dat beide woorden beginnen met een hoofdletter.

Getal Naast tekst (de string) kennen we ook getallen. Dit is heel handig als je iets wilt uitrekenen. Hieronder is weergegeven hoe dit in Python eruitziet. Hier hoeft weer geen ” gebruikt te worden. Let op dat bij komma getallen een punt gebruikt wordt.

```
a = 5
b = 1.3
c = 100
```

Opdracht
Geef de types van onderstaande variabelen. Kies uit: *string*, *getal* of *boolean*.

```
a = 5
b = "hoi"
c = True
```

Code

```
print("het type van variabel a is ...")
print("het type van variabel b is ...")
print("het type van variabel c is ...")
```

[2.0.1] Expressies*

Expressies Met alleen waardes die je direct intypt kan je nog niet hele interessante programmas maken. Als je bijvoorbeeld een sommetje hebt, waarom zou je deze uit moeten rekenen voor de computer? De computer is immers een grote rekenmachine en kan dit voor je doen. Hieronder een voorbeeld hoe dit in Python eruitziet.

```
a = 5*5
b = 2+a
```

Na uitvoeren van het programma zal variabel a de waarde 25 hebben en b de waarde 27. Zo'n sommetje (bijvoorbeeld 5*5) noemen we ook wel een *expressie* in de informatica. Een expressie levert een waarde op (ofwel wordt uitgerekend) en heeft net als een variabel ook een type. Zo levert 5*5 de waarde 25 op en heeft als type getal.

Opdracht
Geef van onderstaande variabelen de waarden en de type. Kies voor type uit: *string*, *getal* of *boolean*.

```
a = 5*2
b = a+1
c = a-b
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
```

[2.0.2] Expressies vervolg*

Wiskundige operaties Zoals je in de voorgaande opdracht gezien hebt kan je gebruik maken van de wiskundige *operaties* (een operatie is bijvoorbeeld plus, min en vermenigvuldigen) die je al kent. Hieronder staat een lijstje van operaties die je allemaal kunt gebruiken bij getallen.

- + **optellen** *voorbeeld:* 1+1 *waarde:* 2
- - **aftrekken** *voorbeeld:* 2-1 *waarde:* 1
- * **vermenigvuldigen** *voorbeeld:* 2*5 *waarde:* 10
- / **delen** *voorbeeld:* 15/5 *waarde:* 3
- ** **machtsverheffing** *voorbeeld:* 2**4 *waarde:* 16
- % **modulo (rest)** *voorbeeld:* 16%5 *waarde:* 1

Boolean operaties Naast de wiskundige operaties die je al kent, heb je ook booleaanse operaties. Deze werken op booleaanse waardes (waar en onwaar). De booleaanse operaties die je kunt gebruiken zijn *and* (en), *or* (of) en *not* (niet).

Het figuur hieronder laat de resultaten van al deze operaties zien.

Boolean operator	Boolean operation	Result
and	False and False	False
	True and True	True
	True and False	False
or	False or False	False
	True or True	True
	True or False	True
not	not True	False
	not False	True

Opdracht
Geef van onderstaande variabelen de waarden en de type. Kies voor type uit: *string*, *getal* of *boolean*.

```
a = 1+3
b = False and True
c = 4/2
d = False or True
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[2.0.3] Tip*

Tip Inmiddels heb je al heel wat operaties gezien, maar er zijn er nog meer. Die zullen bij de volgende opdracht uitgelegd worden. Maak je niet druk dat je nog niet alle operaties in een keer kent. Je zult er genoeg mee oefenen in de komende opdrachten. Mocht het zo zijn dat je niet weet wat een bepaalde operatie doet, kan je het altijd uitproberen door het uit te voeren.

Voorbeeld: Stel je weet niet meer wat de *and* operatie doet en je moet de vraag beantwoorden wat de waarde van *True and True* is. Dit kan je achterhalen door de computer dit voor je te laten berekenen. Doe dit door de expressie te laten weergeven op het Uitvoerscherm door onderstaande code te gebruiken. Zoals je ziet hoef je de waarde niet eerst in een variabel te stoppen.

```
print(True and True)
```

Nadat je op run klikt, zal de computer het resultaat van deze expressie weergeven op het scherm.

Opdracht
Druk op run en kijk wat het resultaat is van deze expressie.

Code

```
1 print(True and True)
```

[2.0.4] Meer expressies*

Vergelijkingsoperaties Naast de wiskundige operaties kan je ook vergelijkings operaties doen voor getallen. Deze zijn hieronder weergegeven.

- == **is gelijk aan** *voorbeeld:* 1==1 *waarde:* True
- != **is ongelijk aan** *voorbeeld:* 1!=1 *waarde:* False
- > **is groter dan** *voorbeeld:* 5>3 *waarde:* True
- < **is kleiner dan** *voorbeeld:* 4<3 *waarde:* True
- >= **is groter of gelijk aan** *voorbeeld:* 5>=5 *waarde:* True
- <= **is kleiner of gelijk aan** *voorbeeld:* 5<=6 *waarde:* False

Het verschil met de wiskundige operaties is dat wiskundige operaties als resultaat type een getal opleveren en vergelijkings operaties als resultaat type een boolean.

String operaties Naast operaties voor getallen en booleans, zijn er ook operaties voor strings. Deze zijn hieronder weergegeven.

- == **is gelijk aan** *voorbeeld:* “a”==“a” *waarde:* True
- != **is ongelijk aan** *voorbeeld:* “a”!=“b” *waarde:* True
- + **aan elkaar plakken** *voorbeeld:* “a”+“b” *waarde:* “ab”

Opdracht
Geef van onderstaande variabelen de waarden en de type. Kies voor type uit: *string*, *getal* of *boolean*.

```
1 a = "a"+"b"
  b = 5<4
3 c = 4**2
  d = "a"=="b"
```

Hoe antwoorden: denk je dat het type van a string is en waarde *antwoord* heeft, vul dan onderstaande code in de Code Editor.

```
print("variabel a heeft als waarde antwoord en als type string")
```

Code

```
1 print("variabel a heeft als waarde ... en als type ...")
  print("variabel b heeft als waarde ... en als type ...")
3 print("variabel c heeft als waarde ... en als type ...")
  print("variabel d heeft als waarde ... en als type ...")
```

[2.0.5] Hergebruik

Opdracht
Geef van onderstaande variabel de waarden en de type. Kies voor type uit: *string*, *getal* of *boolean*.

```
a = "dit"
a = a+"dat"
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
```

[2.0.6] Logica

Opdracht
Geef van onderstaande variabelen de waarden en de type. Kies voor type uit: *string*, *getal* of *boolean*.

```
a = True
b = False
c = a and b
d = a and a
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[2.0.7] Expressieceptie

Opdracht
Geef van onderstaande variabelen de waarden en de type. Kies voor type uit: *string*, *getal* of *boolean*.

```
a = "kwik"+" kwek" +" kwak"
b = 1+2+3
c = 4==(1+3)
d = c==4
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[2.0.8] Tautologie en Tegenspraak

Opdracht
Geef van onderstaande variabelen de waarden en de type. Kies voor type uit: *string*, *getal* of *boolean*.

```
a = 5>6 and 5<6
b = True or 1-1>1
c = 1<0 or 1>0
d = "dit"!=dat and "dit"=="dat"
```

Tip: Mocht je niet meer weten wat de *and* en *or* operaties doen, kijk bij opdracht [2.0.2] voor uitleg hierover.

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[2.0.9] Hergebruik 2

Opdracht
Geef van onderstaande variabelen de waarden en de type. Kies voor type uit: *string*, *getal* of *boolean*.

```
a = "dit"
a = a+a
b = "waarde:"+"4"
c = b + "2"
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
```

[2.0.10] Verwarrend

Opdracht
Geef van onderstaande variabelen de waarden en de type. Kies voor type uit: *string*, *getal* of *boolean*.

```
a = "a"
b = "b"
c = a+b+a
d = (c+"2") != True
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[2.0.11] Logica 2

Opdracht
Geef van onderstaande variabelen de waarden en de type. Kies voor type uit: *string*, *getal* of *boolean*.

```
a = True
b = False
c = (a and b) or a
d = a or (a and b)
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[2.1.0] Python Types*

Types in Python Python biedt de mogelijkheid om het type van een variabel op te vragen. Dit doe je door *type()* te typen. VVan wat tussen de haakjes staat wordt bepaald wat voor type het heeft.. Hieronder een voorbeeld van hoe je het type van een variabel a weergeeft.

```
a = "hoi"
print(type(a))
```

Dit programma geeft onderstaande weer. Wat tussen enkele aanhalingstekens staat geeft de afkorting van het type aan, in dit geval dus *str* (lees stukje hieronder voor afkortingen). Wat *class* hier betekent is niet relevant.

```
<class 'str'>
```

Tot nu toe hebben we gewerkt met de types *boolean*, *getal* en *string*. Python gebruikt deze ook, alleen splitst Python *getal* in twee types: *integer* (gehele getallen zoals 1) en *float* (decimale getallen zoals 1.1). Verder gebruikt Python afkortingen voor deze types. De afkortingen zijn hieronder weergegeven.

bool **boolean**

int **integer**

float **float**

str **string**

Opdracht
Pas het programma aan door op de plek van ... een operator (+, -, ==, !=, etc.) neer te zetten zodat de variabel a het type *boolean* heeft.

Code

```
a = "dit" ... "dat"
print(type(a))
```

[2.1.1] Variabel niet nodig

Opdracht
Pas het programma aan door op de plek van ... een operator (+, -, ==, !=, etc.) neer te zetten zodat deze het type *float* weergeeft.

Code

```
print(type(2 ... 1.5))
```

[2.1.2] Geheel Getal

Opdracht
Pas het programma aan door op de plek van ... een operator (+, -, ==, !=, etc.) neer te zetten zodat de variabel a het type *integer* heeft.

Code

```
a = 5 ... 3
print(type(a))
```

[2.1.3] Breuken

Opdracht
Pas het programma aan door op de plek van ... een operator (+, -, ==, !=, etc.) neer te zetten zodat de variabel a het type *float* heeft.

Code

```
a = 5 ... 3
print(type(a))
```

[2.1.4] Woorden

Opdracht
Pas het programma aan door op de plek van ... een operator (+, -, ==, !=, etc.) neer te zetten zodat de variabel a het type *string* heeft.

Code

```
a = "dit" ... "dat"
print(type(a))
```

[2.1.5] Complex

Opdracht
Pas het programma aan door op de plek van ... een operator (+, -, ==, !=, etc.) neer te zetten zodat de variabel a het type *boolean* heeft.

Code

```
a = 5 + 10 ... 15
print(type(a))
```

[2.2.0] Grote Getallen+

Er wordt aan je gevraagd of 2^128 gelijk is aan 4^64. Voor een mens is dit moeilijk uit te rekenen, voor een computer niet.

Opdracht
Schrijf een programma dat eerst de waarde van 2^128 weergeeft, daarna de waarde van 4^64 en als laatste of deze getallen gelijk aan elkaar zijn.

Voorbeeld: Stel 2^128 is 10 en 4^64 is 12, dan is de uitvoer van je programma zoals hieronder weergegeven.

```
10
12
False
```

Tip: Voor machtsverheffing gebruik je in Python niet ^maar een ander teken. [2.0.2] legt de wiskundige operaties uit.

Code

```

```

A.6.4 Input en casting

[3.0.0] Input*

Tot nu toe hebben we alleen ons programma iets laten weergeven. Interessanter is een programma dat iets aan de gebruiker vraagt en daarmee iets doet. Maar hoe doe je dit in Python?

Met *input()* wordt een waarde gevraagd aan de gebruiker. Deze moet de gebruiker invoeren op het Uitvoerscherm en vervolgens op *enter* drukken. Net als bij *print()* kan je tussen de haakjes een waarde zetten. Deze wordt dan weergegeven op het Uitvoerscherm. Een voorbeeld van een programma dat vraagt de gebruiker een waarde te geven is hieronder weergegeven.

```
input("geef een waarde")
```

Opdracht
Neem onderstaande code over in de Code Editor en druk op run. Klik vervolgens op submit om de opdracht in te leveren.

```
a = input("geef een waarde")
print(a)
```

Code

```

```

[3.0.1] Optelsom*

Opdracht
Wat geeft onderstaande programma weer op het Uitvoerscherm wanneer een gebruiker eerst 1 invult en vervolgens 2? De uitleg waarom het programma dit weergeeft vind je bij de volgende opdracht.

```
a = input("geef een waarde")
b = input("geef een waarde")
print(a+b)
```

Code

```
print("dit programma geeft ... weer")
```

[3.0.2] Casting*

Waarschijnlijk had je bij de vorige opdracht verwacht dat het programma 3 zou weergeven. Nu gaf het 12 weer, maar waarom?

Dit komt omdat de waarde die *input()* oplevert van het type *string* is. Maar wat nou als de optelsom wil doen in plaats van de tekst aan elkaar plakken?

Casten Hiervoor is er *casten*. Met casten verander je het type van een variabel of expressie. Zo kan je van een string een getal maken (geheel of decimaal) en van een getal een string. Hieronder staat weergegeven hoe je dit kan doen.

str() **naar string** *voorbeeld:* str(10) *waarde:* “10”

int() **naar geheel getal** *voorbeeld:* int(“12”) *waarde:* 12

float() **naar decimaal getal** *voorbeeld:* float(“12.3”) *waarde:* 12.3

Het is ook mogelijk om van een float een integer te maken. *Voorbeeld: int(12.99) waarde: 12*. Zoals je ziet wordt het decimale deel dan weggelaten.

Opdracht
Geef van onderstaande variabelen de waarden en de type. Kies voor type uit: *string*, *integer*, *float* of *boolean*.

```
a = str(True)
b = a + "False"
c = "10"
d = int(c) + 5
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[3.0.3] Mix and Match*

Tot nu toe hebben we vooral expressies gezien die alleen tussen hetzelfde type gaan (bijvoorbeeld getal + getal en string + string). Nu kan je ook code schrijven wat een operatie tussen twee verschillende types doet, bijvoorbeeld *"10" + 1*. De vraag is nu wat dit voor waarde op zou moeten leveren? Gezien hier geen eenduidig antwoord op is, is er in Python voor gekozen om in dit geval een error terug te geven. Het Uitvoerscherm geeft dan onderstaande weer.

```
Traceback (most recent call last):
File "python", line 1, in <module>
TypeError: must be str, not int
```

In dit bericht kunnen we op de onderste regel lezen wat voor fout er precies in onze code zit. Python geeft aan dat we een *TypeError* hebben. Dit betekent dat we een operatie proberen te doen die niet mogelijk is met de gegeven types (in dit geval proberen we optellen (de operatie) te doen met een string en integer).

Zoals je dus ziet is het niet mogelijk om de operatie + tussen twee verschillende types te zetten. Dit geldt voor alle operaties op twee na: == en !=. *Bijvoorbeeld: 2 == "2"* levert *False* op.

Verder zijn alle wiskundige operaties (+, -, *, etc.) mogelijk tussen alle getallen. Hier mag dus integers en floats gemixt worden (*bijvoorbeeld: 1.1*2* levert de waarde *2.2* op en is van het type float). Let hierbij wel op dat het resultaat tussen een float en integer altijd een float is.

Opdracht
Geef van onderstaande variabelen de waarden en de type of als deze een error geeft. Kies voor type uit: *string*, *integer*, *float* of *boolean*. In het geval dat een variabel toewijzing een error oplevert, kies als waarde en type *error*.

```
a = 10.5 - 5
b = 5 - 1.0
c = str(8) + 2
d = "16" + 4
```

Hoe antwoorden: Stel je denkt dat de toewijzing aan variabel a niet kan en een error oplevert. Vul dan onderstaande code in de Code Editor

```
print("variabel a heeft als waarde error en als type error")
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[3.0.4] Cast in expressie

Opdracht
Geef van onderstaande variabelen de waarden en de type of als deze een error geeft. Kies voor type uit: *string*, *integer*, *float* of *boolean*. In het geval dat een variabel toewijzing een error oplevert, kies als waarde en type *error*.

```
a = float(5) - 5
b = int(5.5)
c = int(5/2)
d = 5/2.5
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[3.0.5] Tafels

Opdracht
Geef van onderstaande variabelen de waarden en de type of als deze een error geeft. Kies voor type uit: *string*, *integer*, *float* of *boolean*. In het geval dat een variabel toewijzing een error oplevert, kies als waarde en type *error*.

```
a = "5*5="+25
b = "5*6="+str(30)
c = "5*7="+5*7
d = "5*7="+str(5*7)
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[3.0.6] Castception

Opdracht
Geef van onderstaande variabelen de waarden en de type of als deze een error geeft. Kies voor type uit: *string*, *integer*, *float* of *boolean*. In het geval dat een variabel toewijzing een error oplevert, kies als waarde en type *error*.

```
a = str(int(10.5) + 5)
b = float(10+int("12"))
c = True+"self"
d = str(True)+"self"
```

Code

```
print("variabel a heeft als waarde ... en als type ...")
print("variabel b heeft als waarde ... en als type ...")
print("variabel c heeft als waarde ... en als type ...")
print("variabel d heeft als waarde ... en als type ...")
```

[3.1.0] Doorzetter*

Zoals je gemerkt hebt wordt een programma altijd van boven naar beneden uitgevoerd. Python ontdekt alleen een TypeError wanneer deze uitgevoerd wordt. Dat betekent dat wanneer je code schrijft waar op de laatste regel een TypeError zit, alle bovenstaande regels eerst nog wel worden uitgevoerd en onderstaande niet.

Opdracht
Voer het programma uit, kijk dat inderdaad a en b weergegeven worden. Pas het programma aan zodat het geen TypeError meer heeft en onderstaande output geeft.

```
a
b
c4
d
```

Code

```
print("a")
print("b")
print("c"+4)
print("d")
```

[3.1.1] Vraagje

Opdracht
Pas het programma aan zodat het de volgende vraag stelt aan de gebruiker *geef input*.

Code

```
a = input()
print(a)
```

[3.1.2] Yet another TypeError

Opdracht
Pas het programma aan zodat het geen TypeError meer heeft en onderstaande output geeft. Maak gebruik van casting,

```
waarde van a=16
```

Code

```
a = 4*4
print("waarde van a="+a)
```

[3.1.3] Verandering

Opdracht
Pas het programma aan door casting te gebruiken zodat de variabel a het type *string* en waarde 10.0 heeft. Laat de laatste regel onveranderd.

Code

```
a = float("5") * 2
print("waarde=" + str(a) + ", "+ "type=" + str(type(a)))
```

[3.1.4] Vraagje 2

Opdracht
Pas het programma aan zodat het een waarde van de gebruiker vraagt met het bericht *geef een waarde*. Laat de laatste regel onveranderd.

Code

```
a = ...
print("waarde=" + str(a) + ", "+ "type=" + str(type(a)))
```

[3.1.5] Verandering 2

Opdracht
Pas het programma aan door casting te gebruiken zodat de variabel a het type *float* en waarde 12.0 heeft. Laat de laatste regel onveranderd.

Code

```
a = 10 + 2
print("waarde=" + str(a) + ", "+ "type=" + str(type(a)))
```

[3.1.6] Verandering 3

Opdracht
Pas het programma aan door casting te gebruiken zodat de variabel a het type *integer* en waarde 1 heeft. Laat de laatste regel onveranderd.

Code

```
a = 1.2
print("waarde=" + str(a) + ", "+ "type=" + str(type(a)))
```

[3.2.0] Hallo jij!+

Opdracht
Schrijf een programma dat aan de gebruiker input vraagt met de volgende vraag *wat is je naam?*. Het programma moet vervolgens *hallo <ingevulde naam>!* weergeven.

Voorbeeld: Stel je typt *bob* in als naam, dan geeft het programma de onderstaande uitvoer.

```
hallo bob!
```

Code

```

```

[3.2.1] Rekenmachine+

Opdracht
Schrijf een programma dat twee decimale getallen vraagt aan de gebruiker en vervolgens de som van beide getallen weergeeft.

Voorbeeld: Stel als eerste getal wordt 5 gegeven en als tweede getal 6.5, dan geeft het programma onderstaande weer

```
11.5
```

Code

```

```

[3.2.2] BMI+

BMI staat voor Body Mass Index en geeft de verhouding tussen lengte en gewicht van een persoon weer. BMI kan als volgt berekend worden

```
bmi = gewicht/(lengte^2)
```

Hier is gewicht in kg en lengte in m.

Opdracht
Schrijf een programma wat eerst het gewicht (in kg) van een gebruiker en vervolgens de lengte (in m) vraagt. Hierna moet de BMI van deze gebruiker weergegeven worden.

Voorbeeld: Stel een gebruiker vult 2 als lengte en 80 als gewicht in, dan geeft het programma onderstaande weer

```
20.0
```

Code

```
1
```

[3.2.3] Korting+

Wanneer je bijvoorbeeld kleding aan het kopen bent zie je vaak aanbiedingen waarop de originele prijs staat en een percentage korting wat je krijgt. Dit is niet altijd even makkelijk uit te rekenen uit je hoofd.

Opdracht
Schrijf een programma wat eerst het originele bedrag (in euros) en daarna de korting in procenten vraagt aan de gebruiker. Vervolgens moet het programma weergeven wat het originele bedrag was, de hoeveelheid korting gekregen en het bedrag dat je moet betalen.

Voorbeeld 1: Stel een item kost 50 euro en je krijgt 10 procent korting. Het programma geeft dan onderstaande weer

```
1 bedrag was 50.0 euro, korting is 5.0 euro, nieuwe bedrag is 45.0 euro
```

Voorbeeld 2: Stel een item kost 100 euro en je krijgt 25 procent korting. Het programma geeft dan onderstaande weer

```
1 bedrag was 100.0 euro, korting is 25.0 euro, nieuwe bedrag is 75.0 euro
```

Code

```
1
```

A.6.5 Keuze

[4.0.0] Keuze*

Tot nu toe hebben we alleen programmas geschreven waarvan de uitkomst altijd eenzelfde pad in de code volgt (van boven naar beneden). Soms is het nodig om de code een ander pad te laten volgen basis van een conditie. Neem als voorbeeld de BMI rekenmachine die je gemaakt hebt. Deze geeft nu alleen maar de waarde van de BMI aan. Een leuke uitbreiding zou zijn dat dit programma te laag, goed of te hoog zou aangeven. In dat geval verandert de uitvoer die het programma geeft op basis van de waarde van de BMI. Deze uitbreiding van de BMI rekenmachine zal je maken bij opdracht [4.2.2] BMI 2.

if-statement Met behulp van een *if-statement* kan je bepaalde code wel of niet laten uitvoeren op basis van een bepaalde conditie. In Python ziet dit er als volgt uit.

```
1 if <conditie>:
  __print("hoi")
```

In dit voorbeeld moet *<conditie>* vervangen worden door een expressie van het type boolean. Op het moment dat deze expressie de waarde True (waar) heeft wordt de code die bij de if-statement hoort uitgevoerd. Als de waarde van deze expressie False is, wordt de code die bij de if-statement hoort niet uitgevoerd. Hoe je dit kan lezen is als de conditite waar is dan voer de bijbehorende code uit.

Welke code hoort bij de if-statement en welke bij het “normale” programma? Het helpt om hier stukken code als bij elkaar horende blokken te zien. Tot nu toe heb je alleen nog maar programmas gezien die uit 1 blok bestaan. Blokken worden in Python bepaald door de hoeveelheid spaties die ervoor staan. Hebben regels dezelfde hoeveelheid spaties ervoor staan, dan horen die bij elkaar in een blok. Dit geldt alleen voor achtereenvolgende regels! Voor een blok staan altijd 2 of een meervoud van 2 aan spaties. Verder kun je alle code wat je geleerd hebt (en nog zal leren) gebruiken in een blok.

Een programma is gegeven wat de blokken duidelijker maakt. Zoals je ziet begint het blok code dat bij de if-statement hoort met 2 spaties meer dan de if-statement zelf. Op deze manier weet Python dat dat blok code bij de if-statement hoort.

Opdracht
Voer het programma uit en druk op submit.

Code

```
if True:
__print("deze regel hoort bij blok 1")
__print("deze regel hoort ook bij blok 1")
print("deze regel begint een nieuw blok 2")
print("deze regel hoort bij blok 2")
if False:
__print("deze regel begint weer een nieuw blok en wordt niet uitgevoerd")
```

[4.0.1] Anders en anders-als*

Je hebt nu gezien dat code afhankelijk van een conditie wel of niet uitgevoerd wordt. Er zijn nog wat meer uitbreidingen op de if-statements. Deze zijn hieronder uitgelegd en zal je in de komende opdrachten mee oefenen. Wees dus niet bang dat je niet meteen alles snapt en de verschillen tussen de verschillende uitbreidingen helder hebt.

Anders Soms zal je in je programma twee soorten gedrag willen hebben: wanneer een conditie wel waar is en wanneer deze onwaar is. Hieronder is een programma gegeven die aan de gebruiker een getal vraagt. Wanneer de ingevoerde waarde groter is dan 0 dan geeft die *positief* weer. Is dit niet het geval dan geeft het programma *negatief* weer.

```
a = float(input("geef een getal"))
if a>0:
__print("positief")
else:
__print("negatief")
```

Zoals je ziet zorgt het woord *else* (vertaald naar Nederlands: anders) ervoor dat wanneer *a>0* niet waar is, *print("negatief")* wordt uitgevoerd. Hoe je deze code kan lezen is: als a groter is dan 0 dan geef *positief* weer anders geef *negatief* weer.

anders-als De laatste uitbreiding is anders-als. Neem weer als voorbeeld het programma dat positief en negatief weergeeft. Op het moment dat een gebruiker 0 als invoer geeft, zal het programma negatief weergeven. 0 is neutraal. Wanneer de gebruiker 0 als invoer geeft willen we eigenlijk dat het programma *neutraal* weergeeft. Het programma wat dit doet is hieronder weergegeven.

```
a = float(input("geef een getal"))
if a>0:
__print("positief")
elif a<0:
__print("negatief")
else:
__print("neutraal")
```

Zoals je ziet zorgt het woord *elif* (vertaald naar Nederlands: anders-als) ervoor dat er een extra conditie gecontroleerd kan worden. Hoe je deze code kan lezen is: als a groter is dan 0 dan geef *positief* weer anders als a kleiner is dan 0 geef *negatief* weer anders geef *neutraal* weer.

Opdracht
Voer de code uit en probeer verschillende getallen als invoer. Druk op submit om de opdracht in te leveren.

Code

```
a = float(input("geef een getal"))
if a>0:
__print("positief")
elif a<0:
__print("negatief")
else:
__print("neutraal")
```

[4.0.2] Tellen*

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
if a>=0:
__a = a+1
if a>=0:
__a = a+1
if a>=0:
__a = a+1
```

Code

```
print("a heeft als waarde ...")
```

[4.0.3] Tellen 2*

Zoals je in de opdracht hieronder ziet zie je dat een*elif* altijd hoort bij een if-statement. Ook zie je dat deze meerdere keren gebruikt kan worden. Nu is de vraag wat het verschil is met de voorgaande opdracht? Bij elif is het zo dat wanneer 1 conditie van een eerdere if of elif waar is, de rest wordt overgeslagen. Hieronder een code voorbeeld wat dit verduidelijkt. Deze code zal alleen *dit wordt wel uitgevoerd, omdat conditie waar is* weergeven.

```
if False:
__print("dit wordt niet uitgevoerd, omdat conditie onwaar is")
elif True:
__print("dit wordt wel uitgevoerd, omdat conditie waar is")
elif True:
__print("dit wordt niet uitgevoerd, omdat voorgaande if waar is")
```

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
if a>=0:
__a = a+1
elif a>=0:
__a = a+1
elif a>=0:
__a = a+1
```

Code

```
print("a heeft als waarde ...")
```

[4.0.4] Opwarmertje

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 5
if False:
__a = 10
```

Code

```
print("a heeft als waarde ...")
```

[4.0.5] Opwarmertje 2

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 5
b = 10>5
if b:
__a = 10
```

Code

```
print("a heeft als waarde ...")
```

[4.0.6] Lots of Choice

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
if a>=0:
__a = a+1
elif True:
__a = 0
if a>=0:
__a = a+1
elif a>=1:
__a = a+1
```

Code

```
print("a heeft als waarde ...")
```

[4.0.7] ifceptie

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
if a>=0:
__a = a+1
__if a==1:
____a = 0
```

Code

```
print("a heeft als waarde ...")
```

[4.0.8] ifceptie 2

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
if a>=0:
__a = a+1
__if a==1:
____a = 0
__a = 20
```

Code

```
print("a heeft als waarde ...")
```

[4.0.9] Welke if?

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
if a==1:
__a = a+1
__if a==2:
____a = 0
elif a==2:
__a = 5
else:
__a = 10
```

Code

```
print("a heeft als waarde ...")
```

[4.0.10] Whole Lotta elif

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 1
if a==1:
__a = a+1
__if a==1:
____a = 0
__elif a==2:
____a = 5
__elif a==3:
____a = 8
__a = 10
elif a==10:
__a = 5
```

Code

```
print("a heeft als waarde ...")
```

[4.0.11] Onnodig

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 1
if a==1:
__a = a+1
__if a==1:
____a = 0
__elif a==2:
____a = 5
__elif a==3:
____a = 8
a=30
```

Code

```
print("a heeft als waarde ...")
```

[4.0.12] Welke if? 2

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
1  a = 30
   if a/10==3:
3  __if a*3<100:
   ____if a+20<40:
5  ______a = 0
   ____else:
7  ______a = 100
   __else:
9  ____a = 150
   else:
11 __a = 64
```

Code

```
1  print("a heeft als waarde ...")
```

[4.1.0] Output?

Opdracht
Pas het programma aan zodat het *hoi* weergeeft. Doe dit door alleen regel 1 aan te passen.

Code

```
1  if False:
   __print("hoi")
```

[4.1.1] Output? 2

Opdracht
Pas het programma aan zodat het *hoi* weergeeft. Doe dit door alleen regel 1 aan te passen.

Code

```
   if False:
2  __print("hoi")
   else:
4  __print("hey")
```

[4.1.2] Output? 3

Opdracht
Pas het programma aan zodat het *hoi* weergeeft. Doe dit door alleen regel 1 aan te passen.

Code

```
   if True:
2  __print("hi")
   elif True:
4  __print("hoi")
   else:
6  __print("hey")
```

[4.1.3] It's just an if

Opdracht
Pas het programma aan zodat het uiteindelijk 10 weergeeft. Doe dit door alleen regel 1 aan te passen.

Code

```
a = 3
if a==1:
__a = 2
elif a==2:
__a = 3
elif a==3:
__a = 4
elif a==4:
__a = 5
else:
__a = 10 print(a)
```

[4.1.4] Vind de waarde

Opdracht
Pas het programma aan zodat het uiteindelijk 42 weergeeft. Doe dit door alleen regel 1 aan te passen.

Code

```
1  a = 42
   if a>5:
3  __a = 5
   elif a<1:
5  __a = 1
   elif a>4:
7  __a = 4
   elif a<2:
9  __a = 2
   else:
11 __a = 42
   print(a)
```

[4.1.5] el

Opdracht
Pas het programma aan zodat het uiteindelijk 42 weergeeft. Doe dit door alleen regel 8 aan te passen.

Code

```
a = -1
if a>0:
__a = 13
elif a>1:
__a = 1
elif a==-1:
__a = 42
if a>10:
__a = 2
else:
__a = 11
print(a)
```

[4.1.6] The knights who say if

Opdracht
Pas het programma aan zodat het uiteindelijk 0 weergeeft. Doe dit door alleen regel 4 aan te passen.

Code

```
a = 10
if True:
__a = 5
elif a==5:
__a = 0
print(a)
```

[4.2.0] Positief en negatief+

Opdracht
Schrijf een programma dat de gebruiker om een decimaal getal vraagt. Het programma moet vervolgens weergeven of dit getal positief (groter dan 0) of negatief is (kleiner dan 1). In deze opdracht wordt 0 als positief gezien.

Voorbeeld 1: Stel de gebruiker voert -1 in, dan geeft het programma het volgende weer.

```
negatief
```

Voorbeeld 2: Stel de gebruiker voert 0 in, dan geeft het programma het volgende weer.

```
positief
```

Voorbeeld 3: Stel de gebruiker voert 1 in, dan geeft het programma het volgende weer.

```
positief
```

Code

```

```

[4.2.1] Neutraal+

Het vorige programma dat je geschreven hebt is niet gebruiksvriendelijk en correct (0 is niet positief).

Opdracht
Schrijf een programma die de gebruiker om een decimaal getal vraagt. Het programma moet vervolgens weergeven wat het ingevulde getal was en of dit getal positief, negatief of neutraal is.

Voorbeeld 1: Stel de gebruiker voert -1 in, dan geeft het programma het volgende weer.

```
-1 is negatief
```

Voorbeeld 2: Stel de gebruiker voert 0 in, dan geeft het programma het volgende weer.

```
0 is neutraal
```

Voorbeeld 3: Stel de gebruiker voert 1 in, dan geeft het programma het volgende weer.

```
1 is positief
```

Code

```

```

[4.2.2] BMI 2+

Aan de hand van het BMI van een persoon kan ook gezegd worden of deze persoon ondergewicht, overgewicht of normaal gewicht heeft. Bij volwassen personen ligt de grens van normaal gewicht tussen 18.5 en 25. Dit betekent dat alles onder de 18.5 ondergewicht is en alles boven de 25 overgewicht.

Formule voor BMI was.

```
bmi = gewicht/(lengte^2)
```

Hier is gewicht in kg en lengte in m.

Opdracht
Schrijf een programma wat eerst het gewicht (in kg) van een gebruiker en vervolgens de lengte (in m) vraagt. Hierna moet de BMI van deze gebruiker weergegeven worden in combinatie met of deze ondergewicht, overgewicht of normaal gewicht heeft.

Voorbeeld 1: Stel de gebruiker heeft een BMI van 18.1, dan geeft het programma het volgende weer.

```
BMI is 18.1, je hebt ondergewicht
```

Voorbeeld 2: Stel de gebruiker heeft een BMI van 22.4, dan geeft het programma het volgende weer.

```
BMI is 22.4, je hebt normaal gewicht
```

Voorbeeld 3: Stel de gebruiker heeft een BMI van 26.3, dan geeft het programma het volgende weer.

```
BMI is 26.3, je hebt overgewicht
```

Code

```

```

[4.2.3] FizzBuzz+

Opdracht
Schrijf een programma wat een geheel getal vraagt aan de gebruiker. Vervolgens moet er op het moment dat het getal een meervoud is van 3 en 5 *fizzbuzz* weergegeven worden, een meervoud van alleen 3 *fizz* en een meervoud van alleen 5 *buzz*. In alle andere gevallen moet het ingevoerde getal weergegeven worden.

Hint: Om erachter te komen of een getal een meervoud is van een ander getal kan je de modulo (%) operator gebruiken (restant van deling moet gelijk zijn aan 0). *Voorbeeld:* 15 is een meervoud van zowel 3 en 5 (omdat 15%5==0 en 15%3==0). 16 is geen meervoud van 5 en ook niet van 3 (omdat 16%5==1 en 16%3==1).

Voorbeeld 1: Stel de gebruiker vult 9 in, dan geeft het programma het volgende weer.

```
fizz
```

Voorbeeld 2: Stel de gebruiker vult 15 in, dan geeft het programma het volgende weer.

```
fizzbuzz
```

Voorbeeld 3: Stel de gebruiker vult 11 in, dan geeft het programma het volgende weer.

```
11
```

Code

```

```

[4.2.4] BMI 3+

De BMI kent naast onder, over en normaal gewicht meer categorieën. Deze zijn hieronder weergeven.

minder dan 18,5 *ondergewicht*

18,5 tot 25 *normaal gewicht*

25 tot 27 *licht overgewicht*

27 tot 30 *matig overgewicht*

30 tot 40 *ernstig overgewicht*

meer dan 40 *ziekelijk overgewicht*

Opdracht
Schrijf een programma wat eerst het gewicht (in kg) van een gebruiker en vervolgens de lengte (in m) vraagt. Hierna moet de BMI van deze gebruiker weergegeven worden in combinatie met bovengenoemde categorieën.

Voorbeeld 1: Stel de gebruiker heeft een BMI van 28.3, dan geeft het programma het volgende weer.

```
BMI is 28.3, je hebt matig overgewicht
```

Voorbeeld 2: Stel de gebruiker heeft een BMI van 42.1, dan geeft het programma het volgende weer.

```
BMI is 42.1, je hebt ziekelijk overgewicht
```

Code

```

```

A.6.6 Herhaling

[5.0.0] Herhaling*

In het vorige hoofdstuk heb je gezien dat code niet altijd dezelfde route volgt. Naast stukken code wel of niet uitvoeren is het ook mogelijk om stukken code meerdere malen uit te voeren.

Een voorbeeld van een programma waar dit nodig is, is een programma dat strafregels voor je schrijft (deze zal je maken in opdracht [5.2.0]). Wat dit programma moet doen is een gegeven zin een gegeven aantal keer weergeven. Python kent twee manieren om code meerdere malen uit te laten voeren. De eerste manier wordt in deze opdracht uitgelegd, de tweede in de volgende opdracht.

Zolang lus De *while-loop* (zolang lus) wordt gebruikt om code meerdere malen uit te voeren zolang een bepaalde conditie waar is. Hieronder is gegeven hoe dit in code eruitziet.

```
while <conditie>:
__print("hoi")
```

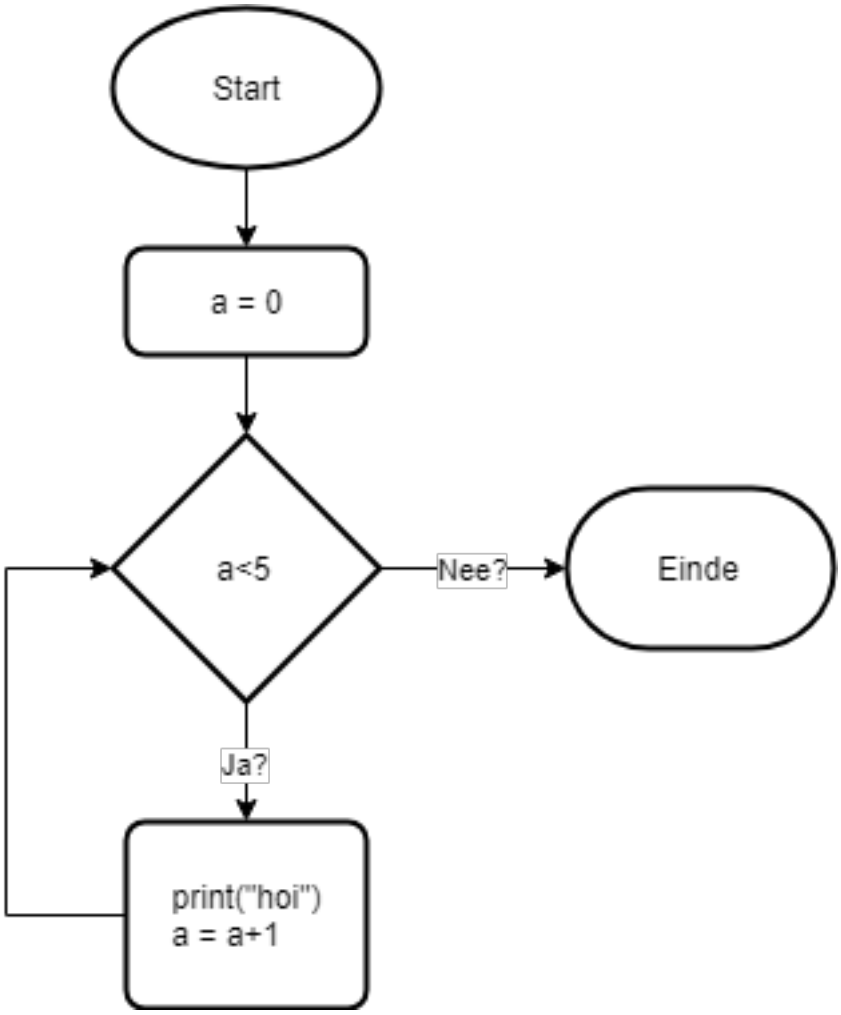
Hier moet, net als bij de if-statement, *<conditie>* vervangen worden door een expressie van het type boolean. Op het moment dat deze expressie onwaar is, wordt de code die bij de while-loop overgeslagen. Is deze waar dan wordt de code uitgevoerd. Op het moment dat al de code in het blok is uitgevoerd, wordt weer teruggesprongen naar de while-loop waar opnieuw gekeken zal worden of de expressie waar is of niet (of de code dus nog een keer moet worden uitgevoerd of niet). Net zoals bij de if-statement hoort, heeft het blok code dat bij de while-loop hoort twee spaties meer ervoor dan de while-loop zelf.

Gegeven is een programma dat zolang a kleiner is dan 5, *hoi* weergeeft. Hoe dit werkt is weergegeven in het plaatje hieronder.
Zoals je kunt zien gaat de code, nadat het blok code van de while-loop is uitgevoerd, weer terug om te kijken of de conditie nog waar is. Is dit waar, dan wordt het blok code weer uitgevoerd. Is dit onwaar, dan wordt het blok code overgeslagen en wordt verder met het programma gegaan.

Opdracht
Voer het programma uit en druk op submit.

Code

```
a = 0
while a<5:
__a=a+1 __print("hoi "+str(a))
```



[5.0.1] Herhaling 2*

Voor-lus De *for-loop* (voor lus) wordt net zoals de zoals lus gebruikt om code meerdere malen uit te voeren. Hieronder is gegeven hoe dit in code eruitziet.

```
1 for i in range(0, 5):
  __print("hoi "+str(i))
```

De for-loop kent een groot verschil met de while-loop. De while-loop heeft een conditie. Zolang die conditie waar is wordt de code uitgevoerd. De for-loop wordt gebruikt om over dingen te *itereren*. Itereren betekent ergens stap voor stap langs gaan. Bijvoorbeeld tellen doe je stap voor stap, je begint bij 1 daarna ga je naar 2 etc.

Met het uitleggen van de for-loop beginnen we met *range(0, 5)*. Dit levert een rij getallen op beginnend bij het eerste ingevulde getal (in dit geval 0) en eindigend net voor het tweede ingevulde getal (in dit geval 5). *range(0, 5)* levert dus de reeks *0, 1, 2, 3, 4* op.

Het blok code dat bij de for-loop hoort wordt vervolgens één keer uitgevoerd voor 0, voor 1, voor 2, voor 3 en voor 4. Nu kan je bij het uitvoeren van de for-loop ook bij de waarde waarvoor die wordt uitgevoerd. Tussen *for* en *in* staat een variabele (in dit geval i). Dit is een normale variabel waar de waarde van de huidige stap (in de informatica ook wel *iteratie* genoemd) wordt opgeslagen. *i* zal bij de eerste keer dat het blok wordt uitgevoerd de waarde 0 hebben, de twee keer de waarde 1, etc. Belangrijk is dat wanneer alle iteraties van de for-loop klaar zijn, je niet meer bij de waarde van *i* kan.

Als je het programma uitvoert zie je dat dit precies hetzelfde doet als het programma in de vorige opdracht. Waarom dan de for-loop gebruiken? Allereerst scheelt het je twee regels code schrijven. Ten tweede zal de for-loop later in de theorie terugkomen en zal je zien dat het voor meer gebruikt wordt. Hier zal ook de term *itereren* duidelijker worden. Maak je dus geen zorgen als je uitleg over de for-loop niet helemaal begrepen hebt.

Opdracht
Voer het programma uit en druk op submit.

Code

```
for i in range(0, 5):
__print("hoi "+str(i))
```

[5.0.2] Nesting*

Zoals al eerder uitgelegd kan je in een blok code alle code gebruiken die je al geleerd hebt (en nog zal leren). Zo kan je bijvoorbeeld if-statements in while- en for-loops gebruiken.

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
while a<3:
__if a==1:
____a = 10
__else:
____a = 20
```

Code

```
print("a heeft als waarde ...")
```

[5.0.3] Tellen

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
for i in range(0,2):
__a = a+1
```

Code

```
print("a heeft als waarde ...")
```

[5.0.4] Nesting 2

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
while a<3:
__for i in range(0,2):
____a = a+1
```

Code

```
print("a heeft als waarde ...")
```

[5.0.5] The Final Count

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
for i in range(0,5):
__a = i
```

Code

```
print("a heeft als waarde ...")
```

[5.0.6] Nog een keertje

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
for i in range(0,5):
__a = a+1
for i in range(0,5):
__a = a+1
```

Code

```
print("a heeft als waarde ...")
```

[5.0.7] Valse start

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
for i in range(2,4):
__a = a+i
```

Code

```
print("a heeft als waarde ...")
```

[5.0.8] Don't Stop Me Now

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 10
b = 0
while b>0:
__b = -1
__a = 100
```

Code

```
print("a heeft als waarde ...")
```

[5.0.9] One or the other

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
b = 0
while a>=b or b>-3:
__if b == -2:
____a = a-10
__a = a-1
__b = b-1
```

Code

```
print("a heeft als waarde ...")
```

[5.0.10] Geen Vooruitgang

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
1 a = 0
  for i in range(2,3):
3 __a = a + i
```

Code

```
print("a heeft als waarde ...")
```

[5.0.11] Stap voor stap

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
1 a = 0
  for i in range(0,4):
3 __i = i+1
  __a = a+i
```

Code

```
1 print("a heeft als waarde ...")
```

[5.0.12] Nestceptie

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
1 a = 0
  for i in range(0,2):
3 __for b in range(0,3):
  ____for c in range(0,2):
5 ______a = a+1
```

Code

```
1 print("a heeft als waarde ...")
```

[5.1.0] Eén tel te weinig

Opdracht
Pas het programma aan zodat onderstaande weergeeft. Doe dit door alleen regel 1 aan te passen.

```
1 0
  1
3 2
```

Code

```
1 for i in range(0,2):
  __print(i)
```

[5.1.1] Een nieuwe start

Opdracht
Pas het programma aan zodat onderstaande weergeeft. Doe dit door alleen regel 1 aan te passen.

```
  1
2 2
```

Code

```
  for i in range(0,3):
2 __print(i)
```

[5.1.2] Verkeerde kant

Opdracht
Pas het programma aan zodat onderstaande weergeeft. Doe dit door alleen regel 2 aan te passen.

```
  1
2 2
  3
```

Code

```
1 a = 0
  while a>4:
3 __a = a+1
  __print(a)
```

[5.1.3] Acht negen tien

Opdracht
Pas het programma aan zodat onderstaande weergeeft. Doe dit door alleen regel 1 aan te passen.

```
  8
2 9
  10
```

Code

```
1 for i in ...:
  __print(i)
```

[5.1.4] Variabel niet gezien

Opdracht
Pas het programma aan zodat onderstaande weergeeft. Doe dit door alleen regel 1 aan te passen.

```
  5
2 6
  7
```

Code

```
1 for ... in range(5, 8):
  __print(a)
```

[5.1.5] Passend maken

Opdracht
Pas het programma aan zodat variabel a de waarde 6. Doe dit door alleen regel 3 aan te passen.

Code

```
a = 0 for i in range(0, 3):
__for ... in range(0, 2):
____a = a+1
print(a)
```

[5.2.0] Strafregels+

In deze opdracht ga je een programma schrijven wat strafregels voor je schrijft.

Opdracht
Schrijf een programma dat eerst om een zin vraagt aan de gebruiker. Vervolgens moet er gevraagd worden hoe vaak deze moet worden weergegeven op het scherm.

Voorbeeld: Stel de gegeven zin is *Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt* en de gegeven hoeveelheid is 10, dan geeft het programma onderstaande weer.

```
Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt
Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt
Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt
Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt
Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt
Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt
Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt
Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt
Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt
Ik zal luisteren naar de docent en stil zijn wanneer hij spreekt
```

Code

```

```

[5.2.1] Tafels+

Opdracht
Schrijf een programma dat vraagt om een getal aan de gebruiker. Vervolgens moet het programma de tafel van dit getal weergeven (zie voorbeeld).

Voorbeeld: Stel het ingevoerde getal is 5. Dan geeft het programma onderstaande weer

```
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

Code

```

```

[5.2.2] Faculteit+

Faculteit in de wiskunde is het product van 1 tot en met het gegeven geheel getal. Faculteit wordt aangegeven met een !. 5! is dus gelijk aan 1*2*3*4*5 = 120.

Opdracht
Schrijf een programma dat de faculteit van een gegeven getal berekend.

Voorbeeld: Stel de gebruiker vult 8 in, dan geeft het programma onderstaande weer

```
40320
```

Code

```

```

[5.2.3] Cumulatieve Rente+

Bij cumulatieve rente berekening wordt rente gerekend over het bedrag plus opgebouwde rente. Stel je hebt 100 euro op de bank en een jaarlijkse rente van 1%. Dan heb je na 1 jaar 1 euro rente. Laat je het nieuwe bedrag (101 euro) op de rekening staan, dan heb je het jaar erop 1,01 euro rente.

Opdracht
Schrijf een programma dat de gebruiker achtereenvolgens vraagt om een startbedrag, de jaarlijkse rente en hoeveel jaar dit bedrag op de rekening blijft staan. Vervolgens geeft het programma het nieuwe bedrag op de rekening weer.

Voorbeeld: Stel de gebruiker vult als bedrag 1000, als percentage 50 en aantal jaar 2. Het programma geeft dan onderstaande weer.

```
13.46972
```

Code

```

```

[5.2.4] Raad het nummer+

Raad het nummer is een spelletje waar er eerst een geheel getal onder de 100 gekozen wordt. Vervolgens moet iemand anders het gekozen getal raden door een getal te geven. Vervolgens zal worden aangeven of dit getal *te hoog*, *te laag* of *goed* is. Op het moment dat het getal goed geraden is, is het spel afgelopen.

Opdracht
Schrijf een programma waar een gebruiker dit spel speelt en het getal 42 moet raden.

Voorbeeld 1: Stel de gebruiker gokt 12 dan geeft het programma onderstaande weer en wordt de gebruiker opnieuw om een cijfer gevraagd.

```
te laag
```

Voorbeeld 2: Stel de gebruiker gokt 42 dan geeft het programma onderstaande weer en vervolgens is het programma klaar met uitvoeren.

```
goed
```

Code

```

```

[5.2.5] Schrikkeljaar+

Een schrikkeljaar is een jaar met 366 dagen. Een jaar is een schrikkeljaar wanneer deze deelbaar is door 4 maar niet door 100 of wanneer deze is door 400. Bijvoorbeeld: 200 is geen schrikkeljaar, maar 400 wel.

Opdracht
Schrijf een programma dat vanaf een gegeven jaartal de komende 20 schrikkeljaren weergeeft (inclusief het gegeven jaartal).

Voorbeeld: Stel de gebruiker vult 2080 in, dan geeft het programma onderstaande weer.

```
2080
2084
2088
2092
2096
2104
2108
2112
2116
2120
2124
2128
2132
2136
2140
2144
2148
2152
2156
2160
```

Code

```

```

[5.2.6] Is het een priemgetal?+

Een priemgetal is een geheel getal wat alleen door 1 en zichzelf deelbaar is. Bijvoorbeeld 5 is alleen deelbaar door 1 en 5, dus een priemgetal. Deze eigenschap speelt een belangrijke rol bij een aantal computer encrypties (versleutelingen). Er is afgesproken dat 1 geen priemgetal is.

Opdracht
Schrijf een programma dat controleert of een gegeven getal een priemgetal is. Het programma geeft *getal is een priemgetal* weer wanneer het gegeven getal priem is en *getal is geen priemgetal* weer wanneer het gegeven getal niet priem is.

Voorbeeld: Stel de gebruiker vult 21 in, dan geeft het programma onderstaande weer.

```
getal is geen priemgetal
```

Code

```

```

[5.2.7] Het a-relaxte optelspel+

Het a-relaxte optelspel is een populair studentenspel waarbij het doel is om gezamenlijk tot de 21 te tellen. De eerste speler die aan de beurt is mag bepalen of er bij 0 of 1 begonnen wordt. Als de 21 gehaald wordt, begint het spel opieuw en mag de speler die 21 haalt een nieuwe regel verzinnen waarmee vervolgens verder gespeeld wordt.

Een beurt bestaat uit het noemen van 1, 2 of 3 getallen. Deze moeten verder gaan waar de voorgaande speler in de reeks gebleven was. Verder mag deze niet hetzelfde aantal getallen noemen als de voorganger.

Daarnaast moet er voor de meervouden van 7 en de getallen waar een 7 in voorkomt (in dit spel zijn dit de getallen: 7, 14, 17 en 21) *juf* gezegd worden in plaats van het getal. Ook moeten getallen voor elk volledig gesloten rondje in het getal een keer extra genoemd worden (1 keer extra: 0, 6, 9, 10, 16, 19, 20 en 2 keer extra: 8, 18).

Opdracht
Schrijf een programma dat de reeks getallen weergeeft wat opgezegd moet worden bij dit spel. Begin bij 0.

Voorbeeld: Hieronder is de reeks tot en met 8 weergegeven(opdracht voor jullie is deze tot en met 21 te maken).

```
0
0
1
2
3
4
5
6
6
juf
8
8
8
```

Code

```
```

[5.2.8] Alle priemgetallen op een stokje+

Opdracht
Schrijf een programma dat vanaf een gegeven getal de komende 20 priemgetallen weergeeft (inclusief het gegeven getal).

Voorbeeld: Stel een gebruiker vult 10 in als getal, dan geeft het programma onderstaande weer.

```
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
```

Code

```
```

A.6.7 Functies

[6.0.0] Functies*

Een functie in programmeren is anders dan een functie zoals je die in de wiskunde hebt leren kennen. Een functie is een blok code die alleen wordt uitgevoerd wanneer deze wordt aangeroepen. Functies hebben verschillende voordelen. Zo zorgt het ervoor dat je code kan hergebruiken en maakt een programma beter leesbaar voor mensen.

Een ander voordeel is dat functies zorgen voor *abstractie*. Abstractie houdt in dat je niet precies de details hoeft de weten, alleen ongeveer wat het doet. Een heel mooi voorbeeld is een functie die je al vaak hebt gebruikt: *print()*. Je weet dat alles wat je tussen haakjes zet, wordt weergegeven. Hoe dit precies gedaan wordt weet je niet.

Functies in Python moeten altijd boven de aanroep in de code geschreven worden. Hieronder is weergegeven hoe dit eruit ziet.

```
def <functienaam>():
__print("hoi")
```

Hier moet *<functienaam>* vervangen worden door een naam van de functie. Net als bij variabelnamen mogen hier geen spaties inzitten en niet beginnen met een cijfer. Net als bij de if-statements, while- en for-loop heeft de functie een blok code. Hier kan je weer alle code die je geleerd hebt gebruiken. De functie op zichzelf doet nog niks. Het moet aangeroepen worden, pas dan wordt het blok code die bij de functie hoort uitgevoerd. Dit doe je door functienaam plus haakjes in de code neer te zetten. Een programma is gegeven die weergeeft hoe dit er uitziet.

Opdracht
Voer het programma uit en druk op submit.

Code

```
def hallo():
__print("Hallo Wereld")

hallo()
```

[6.0.1] Return*

Naast dat een functie alleen iets doet, kan het ook een waarde opleveren. Dit kan worden gedaan door het woord *return* te gebruiken. Python weet dan dat de waarde die achter *return* staat teruggegeven moet worden aan de aanroeper. Een programma is gegeven wat dit weergeeft.

Opdracht
Voer het programma uit en druk op submit.

Code

```
def random():
__a = 4
__return a

print(random())
```

[6.0.2] Scoping*

Een functie kent zijn eigen variabelen. Dat betekent dat variabelen gebruikt en verandert binnen een functie, binnen deze functie blijven. Een programma is gegeven wat dit weergeeft.

Opdracht
Voer het programma uit en druk op submit.

Code

```
a = 5
def random():
__a = 10

print(a)
```

[6.0.3] Parameters en argumenten*

Soms hebben functies iets nodig om hun werk te kunnen doen. Bijvoorbeeld de *print()* functie heeft iets nodig om weer te geven op het uitvoerscherm. Dit iets noemen we een *argument*. Om dit te laten werken moet de functie wel hebben aangegeven dat deze iets verwacht. Dat noemen we een *parameter*. Deze parameter kan gebruikt worden in de functie, maar buiten de functie niet. Een programma is gegeven wat dit weergeeft.

Opdracht
Voer het programma uit en druk op submit.

Code

```
def f(parameter):
__print(parameter)

argument = 10
f(argument)
```

[6.0.4] Meer parameters*

Het is mogelijk meerdere parameters bij een functie te hebben. Hieronder is een programma weergeven die al zijn parameters weergeeft. Deze wordt vervolgens aangeroepen met de waarden “a”,“b” en “c”.

```
def f(parameter1, parameter2, parameter3):
__print(parameter1)
__print(parameter2)
__print(parameter3)

f("a", "b", "c")
```

Zoals je ziet worden parameters gescheiden met een komma. Je bent vrij in het kiezen van de namen zolang deze de regels van de variabelnamen volgt (geen spaties en niet beginnen met een cijfer). Verder is de volgorde waarin je de argumenten noemt belangrijk. Zo krijgt in dit voorbeeld parameter1 de waarde “a”, parameter2 de waarde “b” en parameter2 de waarde “c”.

Opdracht
Voer het programma uit en druk op submit.

Code

```
def f(parameter1, parameter2, parameter3):
__print(parameter1)
__print(parameter2)
__print(parameter3)

f("a", "b", "c")
```

[6.0.5] No return after return*

Na een return in een functie wordt de code die hierna in de functie zou moeten worden uitgevoerd, niet meer uitgevoerd. Een programma is gegeven die dit illustreert.

Opdracht
Voer het programma uit en druk op submit.

Code

```
def random():
__a = 4
__return a
__print("hoi")

print(random())
```

[6.0.6] Welke waarde

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
def f(waarde):
__waarde = waarde + 10

c = 100
f(c)
a = c
```

Code

```
print("a heeft als waarde ..." )
```

[6.0.7] Veel scopes

Opdracht
Wat is de uiteindelijke waarde van de variabel a (niet de a in de functie *f()*)?

```
a = 12
def f():
__a = 4
```

Code

```
print("a heeft als waarde ..." )
```

[6.0.8] For in functie

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
def f():
__b = 0
__for i in range(0,10):
____b = b+1
__return b

a = f()
```

Code

```
print("a heeft als waarde ..." )
```

[6.0.9] For in functie 2

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
a = 0
def f():
__b = 0
__for i in range(0,5):
____b = a+1
__return b

a = f()
```

Code

```
print("a heeft als waarde ..." )
```

[6.0.10] Is 0

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
def is_nul(getal):
__return getal==0

a = is_nul(5)
```

Code

```
print("a heeft als waarde ..." )
```

[6.0.11] Groter dan

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
def biggest(a, b):
__if a>b: ____return a __return b
a = biggest(10, 2)
```

Code

```
print("a heeft als waarde ..." )
```

[6.0.12] Groter dan 2

Opdracht
Wat is de uiteindelijke waarde van de variabel a?

```
def biggest(a, b):
__if a>b: ____return a __return b
a = biggest(1, 20)
```

Code

```
print("a heeft als waarde ..." )
```

[6.0.13] Functieceptie

Opdracht
Wat is de uiteindelijke waarde van de variabel a (niet de a in de functies)?

```
a = 100
def f():
__a=10

def b():
__a=12
__f()

b()
```

Code

```
print("a heeft als waarde ..." )
```

[6.0.14] Functieceptie 2

Opdracht
Wat is de uiteindelijke waarde van de variabel a (niet de a in de functies)?

```
def f():
__return 10

def b():
__return f()

def c():
__return b()

a = c()
```

Code

```
print("a heeft als waarde ..." )
```

[6.1.0] That's not my name

Opdracht
Pas het programma aan zodat het *hallo wereld* weergeeft. Doe dit door alleen regel 1 aan te passen.

Code

```
1 def f():
  __return "hallo wereld"
3
  print(hallo())
```

[6.1.1] Turn around

Opdracht
Pas het programma aan zodat het *hoi* weergeeft. Doe dit door alleen regel 4 aan te passen en gebruik te maken van functie *f()*.

Code

```
  def f(a, b):
2 __return b

4 a = f("hoi", "hallo")
  print(a)
```

[6.1.2] Het missende woord

Opdracht
Pas het programma aan zodat het *hallo wereld* weergeeft. Doe dit door alleen regel 1 aan te passen.

Code

```
1 ... hallo():
  __return "hallo wereld"
3
  print(hallo())
```

[6.1.3] Gebruik de functie

Opdracht
Pas het programma aan zodat het *hoi* weergeeft. Doe dit door alleen regel 4 aan te passen en gebruik te maken van functie *f()*.

Code

```
def f():
__return "hoi"

a = ...
print(a)
```

[6.1.4] Gebruik de functie 2

Opdracht
Pas het programma aan zodat het *hallo* weergeeft. Doe dit door alleen regel 4 aan te passen en gebruik te maken van functie *f()*.

Code

```
1 def f(h):
  __return h
3
  a = ...
5 print(a)
```

[6.1.5] No return

Opdracht
Pas het programma aan zodat het *hello* weergeeft. Doe dit door alleen regel 2 aan te passen.

Code

```
1 def f():
  __"hello"
3
  a = f()
5 print(a)
```

[6.2.0] Kleiner+

Opdracht
Schrijf een functie *smaller()* dat twee parameters heeft. De functie moet het kleinste getal van de twee teruggeven (returnen).

Voorbeeld: Als de functie wordt aangeroepen als *smaller(1,2)*, dan moet deze 1 teruggeven.

Code

```
1
```

[6.2.1] Gootste+

Opdracht
Schrijf een functie *biggest()* dat drie parameters heeft. De functie het grootste getal van de drie teruggeven.

Voorbeeld: Stel de functie wordt aangeroepen als *biggest(1,3,2)*, dan moet 3 teruggeven.

Code

```
1
```

[6.2.2] Is het een priemgetal 2+

Opdracht
Schrijf een functie *is_prime()* wat 1 parameter heeft. De functie moet *True* (boolean waarde) teruggeven wanneer het gegeven getal priem is, False (boolean waarde) wanneer deze niet priem is.

Voorbeeld: Stel de functie wordt aangeroepen als *is_prime(5)*, dan moet deze True teruggeven

Code

```
1
```

[6.2.3] Alle priemgetallen op een stokje 2+

Opdracht
Schrijf een programma dat vanaf een gegeven getal de komende 20 priemgetallen weergeeft (inclusief het gegeven getal). Gebruik hierbij de functie *is_prime()* die je in de vorige opdracht geschreven hebt.

Voorbeeld: Stel een gebruiker vult 10 in als getal, dan geeft het programma onderstaande weer.

```
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
```

Code

```
1
```

[6.2.4] Interval+

Gegeven zijn 3 functies. Van deze functies hoef je niet te weten hoe ze werken, negeer dus de code die er staat! Wel moet je van twee functies *interval_start()* en *interval_eind()* weten wat ze doen (deze moet je gebruiken in deze opdracht). In de code staat boven de functie uitgelegd wat de functies doen. Dit is gedaan door *commentaar* te gebruiken. Het enige doel van commentaar is om code voor de mens duidelijker te maken. In Python wordt een regel commentaar gestart door een #. Alles wat na de # staat wordt genegeerd door Python.

Wiskundige interval notatie Bij wiskunde heb je het over intervallen gehad. Hier is een speciale notatie voor. Deze notatie bestaat uit twee getallen gescheiden door een komma. Het getal voor de komma geeft de start van het interval aan en het getal na de komma geeft het einde van het interval aan. Blokhaken ("[" en "]") geven aan of het getal meetelt. Ronde haken ("(" en ")") geven aan of het getal zelf niet meedoet.

Opdracht
Schrijf een programma die een wiskundig interval vraagt aan de gebruiker en alle gehele getallen in dit interval weergeeft. Maak gebruik van de functies *interval_start()* en *interval_einde()*.

Voorbeeld 1: Stel de gebruiker vult $[5, 8]$ in, dan moet het programma onderstaande weergeven.

```
5
6
7
8
```

Voorbeeld 2: Stel de gebruiker vult $(5, 8)$ in, dan moet het programma onderstaande weergeven.

```
6
7
```

Code

```
import re

# functie die van een interval string de inclusieve start en eind bepaald
def interval(i):
__r = re.compile("\d+")
__start, eind = r.findall(i)
__start = int(start)
__eind = int(eind)
__if i[0]=="(":
____start = start + 1
__if i[-1]==")":
____eind = eind - 1
__return start, eind

# functie die van een interval de start bepaald.
# Stel i="[5,8]", dan geeft deze functie 5 terug.
# Stel i="(5,8]", dan geeft deze functie 6 terug.
def interval_start(i):
__return interval(i)[0]

# functie die van een interval het einde bepaald.
# Stel i="[5,8]", dan geeft deze functie 8 terug.
# Stel i="[5,8)", dan geeft deze functie 7 terug.
def interval_eind(i):
__return interval(i)[1]
```

A.6.8 Lijsten

[7.0.0] Lijsten*

Kijk de youtube video over uitleg over lijsten: https://www.youtube.com/watch?v=ohCDWZgNIU0

In deze video worden een aantal dingen genoemd die je nog niet kent (en eerst ook nog niet hoeft te kennen). Het eerst is dat er in de video vaak een vergelijking wordt gedaan met *sets*. Het tweede is de *dir()* functie. Negeer deze dingen voor nu.

Gegeven is een programma die laat zien hoe je een lijst aanmaakt en er waardes instopt. De waardes in een lijst worden ook wel *elementen* genoemd.

Opdracht
Voer het programma uit en druk op submit.

Code

```
l = [1, 2]
l.append(3)
l.append(4)
print(l)
```

[7.0.1] Lengte*

Soms is het handig om te weten hoe lang een gegeven lijst is. Dit kan je doen door de *len()* functie te gebruiken. Een programma is gegeven die laat zien hoe deze functie gebruikt wordt.

Opdracht
Voer het programma uit en druk op submit.

Code

```
l = [1, 2, 3, 4]
print(len(l))
```

[7.0.2] Itereren*

Soms wil je langs alle waardes in een lijst gaan (over een lijst itereren). Hier kunnen we onze grote vriend de for-loop voor gebruiken. Gegeven is een stuk code weergegeven die langs alle waardes gaat van begint tot en met eind.

Opdracht
Voer het programma uit en druk op submit.

Code

```
l = ["kwik", "kwak", "kwek"]
for i in l:
__print(i)
```

[7.0.3] Locaties*

Opdracht
Wat zijn de waardes van de variabelen a, b en c in onderstaande programma?

Code

```
l = [1, 2, 3, 4, 5]
a = l[0]
b = l[2]
c = l[-1]
```

print("de waarde van a is ...)˙ print("de waarde van b is ...)˙ print("de waarde van c is ...")

[7.0.4] Splicing*

Opdracht
Wat zijn de waardes van de variabelen a, b en c in onderstaande programma?

```
l = [1, 2, 3, 4, 5]
a = l[0:3]
b = l[0:1]
c = l[0:-1]
```

Hoe antwoorden: Stel variabel a is van het type lijst en heeft waarde

$$1, 2$$

. Vul dan onderstaande code in de Code Editor (let op de spaties).

```
print("de waarde van a is [1, 2])"
```

Code

```
print("de waarde van a is ...")
print("de waarde van b is ...")
print("de waarde van c is ...")
```

[7.0.5] Is die aanwezig?*

Als je een lijst hebt is het soms handig om te weten of een bepaalde waarde al aanwezig is. Een programma is gegeven die laat zien hoe je controleert of een waarde wel of niet aanwezig is in een lijst.

Opdracht
Voer het programma uit en druk op submit.

Code

```
l = ["kwik", "kwak", "kwek"]
if "kwak" in l:
__print("kwak is aanwezig")

if "donald" not in l:
__print("donald is niet aanwezig")
```

[7.0.6] Strings*

Stiekem heb je al vanaf het begin met lijsten gewerkt. Een string is namelijk ook een lijst (lijst van enkele karakters). Alle functies die je op lijsten kan doen, kan je ook doen met strings.

Opdracht
Wat zijn de waardes van de variabelen a, b en c in onderstaande programma?

```
a = len("hoi")
b = "Bob"[1]
c = "hallo wereld"[6:12]
```

Code

```
print("de waarde van a is ...")
print("de waarde van b is ...")
print("de waarde van c is ...")
```

[7.0.7] Nesting

Opdracht
Wat zijn de waardes van de variabelen a, b en c in onderstaande programma?

```
l = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
a = l[0][0]
b = l[1][2]
c = l[2][0]
```

Code

```
print("de waarde van a is ...")
print("de waarde van b is ...")
print("de waarde van c is ...")
```

[7.0.8] Aan elkaar plakken

Opdracht
Wat is de waarde van de variabel a in onderstaande programma?

```
l1 = [1, 2, 3]
l2 = [3, 4]
a = l1 + l2
```

Code

```
1 print("de waarde van a is ...")
```

[7.0.9] De eersten zullen de laatsten zijn

Opdracht
Wat is de waarde van de variabel a in onderstaande programma?

```
  l = [1, 2, 3]
2 a = []
  for i in l:
4 __a = [i] + a
```

Code

```
  print("de waarde van a is ...")
```

[7.1.0] Maak een lijst

Opdracht
Pas het programma aan zodat het *[1, 2, 3]* weergeeft. Doe dit door alleen regel 1 aan te passen.

Code

```
  a = ...
2 a.append(1)
  a.append(2)
4 a.append(3)
  print(a)
```

[7.1.1] Toevoeging

Opdracht
Pas het programma aan zodat het *["kwik", "kwak", "kwek"]* weergeeft. Doe dit door 1 regel toe te voegen.

Code

```
1 l = ["kwik", "kwak"]
  print(l)
```

[7.1.2] Hiep hiep hoi

Opdracht
Pas het programma aan zodat het onderstaande weergeeft.

```
  hiep
2 hiep
  hoi
```

Code

```
1 l = ...
  for i in l:
3 __print(i)
```

[7.1.3] Spellen

Opdracht
Pas het programma aan zodat het onderstaande weergeeft.

```
h
o
i
```

Code

```
l = "hoi"
... l:
__print(i)
```

[7.2.0] Spellen+

Opdracht
Schrijf een programma dat aan de gebruiker een woord vraagt. Vervolgens moet het programma het woord spellen (elke letter afzonderlijk weergeven).

Voorbeeld: Stel de gebruiker vult *test* in, dan geeft het programma onderstaande weer.

```
t
e
s
t
```

Code

```

```

[7.2.1] Palindroom+

Een palindroom is een woord dat zowel van links naar rechts als van rechts naar links hetzelfde woord spelt. Voorbeelden zijn *pap, legovogel* en *meetsysteem*.

Opdracht
Schrijf een programma dat aan de gebruiker een woord vraagt. Vervolgens moet het programma het volgende weergeven: *palindroom* wanneer het woord een palindroom is en *geen palindroom* wanneer het gegeven woord geen palindroom is.

Voorbeeld: Stel de gebruiker vult *test* in, dan geeft het programma onderstaande weer.

```
geen palindroom
```

Code

```

```

[7.2.2] Boter, kaas en eieren: Het bord+

In de komende opdrachten ga je het spel boter, kaas en eieren maken. In elk van de komende opdrachten zal je een deel van het spel moeten maken. In opdracht [7.2.6] zal je alle opdrachten combineren (door je antwoord van eerdere opdrachten te kopiëren naar deze) om het spel te laten werken. Mocht je het spel niet kennen, kan je bij de volgende link nalezen hoe het spel werkt: https://nl.wikipedia.org/wiki/Boter-kaas-en-eieren

Voor boter, kaas en eieren is een bord nodig om op te spelen. Een bord bestaat altijd uit een 3 bij 3 bord. In onze versie van het spel gaan we een lijst met strings gebruiken om het bord te representeren. Een functie *print_board()* is gegeven die als parameter deze lijst heeft en vervolgens ervoor zorgt dat het bord netjes op het scherm wordt weergegeven.

Opdracht
Schrijf een functie *init_board()* die een lijst van 9 lang teruggeeft. Deze lijst moet gevuld zijn met lege strings (“”). Test je functie door het gemaakte bord te laten weergeven op het uitvoerscherm door *print_board()* aan te roepen. Deze moet onderstaande uitvoer hebben.

```
*************
* ___ * ___ * ___ *
*************
* ___ * ___ * ___ *
*************
* ___ * ___ * ___ *
*************
```

Code

```
def print_board(l):
__board_string = """*************
*  *  *  *
*************
*  *  *  *
*************
*  *  *  *
*************"""
__if len(l) != 9:
____raise ValueError("Gegeven lijst is niet 9 lang")
__l2 = []
__for i in l:
____if i == "":
______l2.append(" ")
____else:
______l2.append(i)
__print(board_string.format(*l2))
```

[7.2.3] Boter, kaas en eieren: Positie leeg+

Een speler kan alleen zijn teken neerzetten waar nog geen teken geplaatst is.
Hoe de locaties van de lijst samenhangen met het bord is hieronder weergegeven. Zoals je ziet zit locatie 0 links bovenin en locatie 8 rechts onderin.

```
*************
*_0_*_1_*_2_*
*************
*_3_*_4_*_5_*
*************
*_6_*_7_*_8_*
*************
```

Opdracht
Schrijf een functie *position_taken()* die aangeeft of een positie op het veld al gevuld is. Deze functie heeft 2 parameters; het bord en een locatie. De functie geeft True (boolean waarde) wanneer de positie al bezet is, False wanneer deze nog leeg is.

Code

```

```

[7.2.4] Boter, kaas en eieren: Gelijk spel+

Een spelletje van boter, kaas en eieren is afgelopen wanneer een van de twee spelers (X en O) gewonnen heeft of wanneer het bord vol is. In deze opdracht ga je een functie schrijven die controleert of het bord vol is.

Opdracht
Schrijf een functie *is_full()* die aangeeft of een bord vol is of niet. De functie heeft als parameter het bord (de lijst met strings). De functie geeft True (boolean waarde) terug wanneer het bord vol is en False wanneer deze nog niet vol is. Het bord is vol wanneer er geen lege strings meer in de lijst staan.

Code

```
1
```

[7.2.5] Boter, kaas en eieren: Winnaar+

De andere conditie voor wanneer het spel is afgelopen is, is wanneer een van de twee spelers gewonnen heeft (ofwel drie op een rij heeft). Dit kan door ofwel een horizontale rij vol te hebben, een verticale rij of een van de twee diagonalen.

Hoe de locaties van de lijst samenhangen met het bord is hieronder weergegeven. Zoals je ziet zit locatie 0 links bovenin en locatie 8 rechts onderin.

```
1  *************
   *_0_*_1_*_2_*
3  *************
   *_3_*_4_*_5_*
5  *************
   *_6_*_7_*_8_*
7  *************
```

Opdracht
Schrijf de functies *check_horizontal()*, *check_vertical()* en *check_diagonal()* die respectievelijk controleren of één van de horizontale, één van de verticale en één van de diagonalen gevuld is met het teken van een speler. Deze functies hebben twee parameters; het bord en het teken van de speler (X of O) die gecontroleerd moet worden. Deze functies geven True (boolean waarde) terug wanneer een rij alleen gevuld is met het teken van de speler en False wanneer dit niet zo is.

Schrijf vervolgens een functie *check_win()* met 2 parameters (het bord en het teken van de speler die gecontroleerd moet worden) die True (boolean warde) teruggeeft wanneer deze speler gewonnen heeft en False wanneer deze speler (nog) niet gewonnen heeft. Maak gebruik van de functies *check_horizontal()*, *check_vertical()* en *check_diagonal()*.

Code

```
1
```

[7.2.6] Boter, kaas en eieren: the game+

Inmiddels heb je alle functies die nodig zijn om dit spel te kunnen spelen geschreven. Het enige wat nog gedaan moet worden is alles samen laten werken.

Opdracht
Kopieer alle functies die je geschreven hebt in de voorgaande boter, kaas en eieren opdrachten naar deze opdracht (je hebt ze nodig).

Programmeer het spel boter, kaas en eieren. Onderstaande stappenplan kan je misschien helpen.

```
1  Zolang nog geen winnaar is of bord nog niet vol is:
   __Geef het bord weer
3  __Zolang huidige speler niet een geldige positie geeft (tussen 0 en 9 en positie niet bezet):
   ____vraag huidige speler om een positie
5  __Plaats teken van huidige speler op gegeven locatie
   __Controleer of speler gewonnen heeft
7  __Als geen winnaar, volgende speler is aan de beurt
   Geef winnaar weer
```

Deze opdracht zal met de hand nagekeken worden, het is dus niet mogelijk om op submit te klikken en direct te kijken of het programma correct is. Denk je dat het spel werkt zoals het hoort (met andere woorden kan je boter, kaas en eieren spelen met jouw programma), druk dan op submit.

Code

```
def print_board(l):
__board_string = """*************
*  *  *  *
*************
*  *  *  *
*************
*  *  *  *
*************"""
__if len(l) != 9:
____raise ValueError("Gegeven lijst is niet 9 lang")
__l2 = []
__for i in l:
____if i == "":
______l2.append(" ")
____else:
______l2.append(i)
__print(board_string.format(*l2))
```

A.6.9 Opdrachten

[8.0.0] klaar?*

Inmiddels heb je alle basiskennis van programmeren gehad. Dit betekent echter niet dat je alles al kent. Er zijn nog veel meer dingen mogelijk in Python. Het mooie is dat al deze dingen al op het internet staan uitgelegd. Met de basiskennis die je nu hebt zou je verder zelf op het internet antwoorden voor bepaalde vraagstukken kunnen vinden. De website https://stackoverflow.com/ is een goede website om antwoorden te vinden.

In dit hoofdstuk vind je verschillende opdrachten die je kan doen om meer te oefenen (en daarmee ook je cijfer te verhogen). Je bent vrij om te kiezen in welke volgorde je ze maakt. Pas wel op dat sommige opdrachten een vervolg op een andere kunnen zijn. Begin in dat geval bij de eerste opdracht.

Opdracht
Druk op submit.

Code

```

```

[8.0.1] Geheimschrift: Coderen+

Soms wil je berichten naar andere mensen kunnen sturen, zonder dat iemand anders deze berichten kunnen lezen. Een voorbeeld van een manier om dit te doen is de Ceasarcijfer (https://nl.wikipedia.org/wiki/Caesarcijfer).

Opdracht
Schrijf een programma dat een zin codeert volgens de Ceasarcijfer. Het programma vraagt eerst naar de rotatie (lees uitleg Ceasarcijfer wikipedia) en vervolgens om een zin. Het programma geeft de gecodeerde zin volgens de gegeven rotatie weer. Gebruik alleen de kleine letters van het alfabet, codeer spaties als spaties.

Voorbeeld: De gebruiker voert 12 en vervolgens de zin *dit is geheim* in, dan geeft het programma onderstaande weer.

```
puf ue sqtquy
```

Tip: Het kan handig zijn om gebruik te maken van de functies *ord()* en *chr()*: https://stackoverflow.com/questions/704152/how-can-i-convert-a-character-to-a-integer-in-python-and-viceversa

Code

```

```

[8.0.2] Geheimschrift: Decoderen+

Op het moment dat je een gecodeert bericht ontvangt wil je deze ook kunnen lezen. Daarvoor moet je het bericht eerst decoderen.

Opdracht
Schrijf een programma dat een bericht gecodeert volgens de Ceasarcijfer decodeert. Het programma vraagt eerst aan de gebruiker in welke rotatie het bericht gecodeert is, waarna het gecodeerde bericht gevraagd wordt. Het programma geeft het gedecodeerde bericht weer.

Voorbeeld: De gebruiker voert 12 en vervolgens de zin *puf ue sqtquy* in, dan geeft het programma onderstaande weer.

```
dit is geheim
```

Code

```

```

[8.0.3] Torens van Hanoi: Het bord+

De Torens van Hanoi is een spel waarbij het doel is alle schijven (die verschillende groottes hebben) opgestapeld te krijgen op de meest rechtse toren. Het spel kent verder twee regels:

- Er mag maar 1 schijf per keer verplaatst worden
- Nooit mag een grotere schijf of een kleinere liggen.

Meer over dit spel valt te vinden op https://nl.wikipedia.org/wiki/Torens_van_Hanoi

In de komende opdrachten ga je dit spel programmeren. Wij gaan uit van een variant waar met 4 schijven gespeeld wordt. Schijven in dit spel zijn een geheel getal. 4 is de grootste schijf, 1 de kleinste. Gegeven is de functie *print_towers()*. Deze functie print de 3 torens met de verschillende schijven. De functie heeft 1 argument: de torens. De torens zijn een lijst van de 3 torens. Hier heeft de meest linkse toren index 0 en meest rechtse index 2.

In dit spel is een toren een lijst. Een lege lijst betekend dat er geen schijven op deze toren liggen. Het eerste element in de lijst geeft de onderste schijf weer. *Bijvoorbeeld:* [4, 3] is een toren waar een schijf van 3 groot op een schijf van 4 groot ligt.

Opdracht
Schrijf een functie *init_board()* die de begintoestand van het bord (alle schijven op de meest linkse toren) teruggeeft (ofwel een lijst met 3 torens waar alle schijven op volgorde op de meest linkse toren zit). Test je functie door *print_board()* aan te roepen met het gemaakte bord. *print_board()* zou onderstaande moeten weergeven.

```
_______==______________________________________________
______====____________________________________________
_____======__________________________________________
____========________________________________________
____Toren_0___________Toren_1___________Toren_2
```

Code

```
om formatting redenen is de code hier niet toegevoegd
```

[8.0.4] Torens van Hanoi: Input+

Het enige wat je nodig hebt als input is een van en naar locatie. Hier zijn de van en naar locaties de index nummers van de torens. In deze opdracht moet je de input van de gebruiker goed kunnen omzetten.

Opdracht
Schrijf een functie *handle_input()* die de input van de gebruiker verwerkt. Deze heeft 1 parameter: de string input van de gebruiker. Input heeft het formaat *<cijfer> <cijfer>*. De functie geeft een *tuple* terug waar het eerste getal van de tuple de van locatie is en het tweede getal de naar locatie is.

Wat een tuple is en hoe je het met functies gebruik kan je vinden https://www.mantidproject.org/Working_With_Functions:_Return_Values

Voorbeeld: Stel de input string van de gebruiker is *0 1*. Dan geeft de functie onderstaande terug

```
(0, 1)
```

Code

```

```

[8.0.5] Torens van Hanoi: Verplaats schijf+

Het verplaatsen van een schijf werkt als volgt: De bovenste schijf wordt van een toren gehaald. Vervolgens wordt deze geplaatst op een andere toren.

Opdracht
Schrijf een functie *make_move()* die een schrijf van 1 toren naar een andere verplaatst. Deze functie heeft drie parameters: de torens, de index van de toren waar een schijf afgehaald wordt en de index van de toren waar de schijf naartoe moet. De functie past de torens lijst aan en geeft niks terug.

Voorbeeld: Stel de gebruiker wil in de beginsituatie een schijf van plek 0 naar 1 verplaatsen. De lijst van torens ziet er dan als volgt uit.

```
[[4, 3, 2], [1], []]
```

Code

```

```

[8.0.6] Torens van Hanoi: Controle+

Op dit moment wordt nog niet gecontroleerd of een verplaatsing mag. Je kan nu een grotere schijf op een kleinere zetten.

Opdracht
Schrijf een functie *is_valid_move()* die controleert of een verplaatsing mag. Deze functie heeft drie parameters: de torens, de index van de toren waar een schijf afgehaald wordt en de index van de toren waar de schijf naartoe moet. De functie geeft True terug (boolean waarde) wanneer de verplaatsing mag en False wanneer deze niet mag.

Code

```

```

[8.0.7] Torens van Hanoi: Winnaar+

Op het moment dat je het voor elkaar krijgt om alle schijven op de meest rechtse toren te krijgen gebeurt er nog niks. Er is nog geen controle of het spel afgelopen is.

Opdracht
Schrijf een functie *check_win()* die controleert of het spel afgelopen is. De functie heeft één parameter: de torens. De functie geeft True terug (boolean waarde) wanneer de verplaatsing mag en False wanneer deze niet mag.

Code

```
1
```

[8.0.8] Torens van Hanoi: The game+

Inmiddels heb je alle functies die nodig zijn om dit spel te kunnen spelen geschreven. Het enige wat nog gedaan moet worden is alles samen laten werken.

Opdracht
Kopieer alle functies die je geschreven hebt in de voorgaande Torens van Hanoi opdrachten naar deze opdracht (je hebt ze nodig).

Programmeer het spel Torens van Hanoi. Onderstaande stappenplan kan je misschien helpen.

```
1 Zolang het spel nog niet gewonnen is:
  __Geef het bord weer
3 __Vraag om input
  __Als het een geldige verplaatsing is:
5 ____verplaats de schijf
  __Anders:
7 ____Geef weer dat de verplaatsing niet mogelijk is
```

Deze opdracht zal met de hand nagekeken worden, het is dus niet mogelijk om op submit te klikken en direct te kijken of het programma correct is. Denk je dat het spel werkt zoals het hoort (met andere woorden kan je Torens van Hanoi spelen met jouw programma), druk dan op submit.

Code

```
1
```