Het effect van misconcepties op diepleren

Een onderzoek naar het effect van misconcepties op diep leren binnen programmeren

Verheijen, Robert P.A.

Award date:
2023

Onderzoek van Onderwijs

Het effect van misconcepties op diepleren

Een onderzoek naar het effect van misconcepties op diep leren binnen programmeren

In opdracht van de Technische Universiteit Eindhoven

R.P.A. Verheijen (1479407)

Vakgebied: informatica

Begeleiders: dr. R. Taconis en prof. dr. J. van der Veen

EME40/EME41/EME42: 30 ec.

15-04-2023

Versie 1.59

Verklaring inzake TU/e Gedragscode Wetenschapsbeoefening in het kader van de Masterscriptie

Ik heb kennis genomen van de TU/e Gedragscode Wetenschapsbeoefening[1].

Hierbij verklaar ik dat mijn Masterscriptie conform de regels van de TU/e Gedragscode Wetenschapsbeoefening tot stand is gekomen.

Datum

28-12-2022

Roepnaam Achternaam

Robert Verheijen

. . . .

Handtekening

. . . .

Lever de ondertekende verklaring in bij de coördinator van Onderzoek van Onderwijs

[1] Zie: http://www.tue.nl/universiteit/over-de-universiteit/integriteit/wetenschappelijke-integriteit/
Hier is ook de Nederlandse Gedragscode Wetenschapsbeoefening van de VSNU te vinden.
Meer informatie over wetenschappelijke integriteit is te vinden op de websites van de TU/e en de VSNU.

Inhoudsopgave

Samenvatting

Na de introductie in 2016 door SLO van een nieuw examenprogramma informatica is het kennisdomein Programmeren in de volgende basis-programmeerconstructies opgedeeld: toekenning, opeenvolging, herhaling en keuze, functies (met parameters), testen en debuggen.
In de praktijk blijkt het dat veel 4 en 5 havo en 4, 5 en 5 vwo-leerlingen met coderen regelmatig fouten of misconcepties[2] maken die ogenschijnlijk invloed hebben op het diep leren.
In dit onderzoek zijn in de literatuur 18 misconcepties gevonden en opgedeeld in de basis-programmeerconstructies. Aan de hand van de in de literatuur gevonden misconcepties en de randvoorwaarden voor diepleren is een lessenserie opgesteld waarin een 30-tal ontwerpeisen zijn verwerkt en 14 misconcepties behandeld worden. Tijdens het uitvoeren van deze lessen en het praktijkgedeelte werden de leerlingen geobserveerd naar eventuele misconcepties. Uit deze observaties zijn nog vijf misconcepties aan het licht gekomen die niet in de literatuur gevonden zijn. Een voorbeeld hiervan is dat leerlingen een programmeertaal interpreteren als een natuurlijke taal in plaats van een logische taal. Zo werd de zin “als een getal kleiner is dan 0 of groter dan 10 …” vertaalt naar:

```
if getal < 0 or > 10:
```

In plaats van

```
if getal <0 or getal > 10:
```

Na de lessen werd een korte anonieme vragenlijst voorgelegd aan de leerlingen. Uit deze vragenlijst en de observaties kan voorzichtig worden geconcludeerd dat leerlingen (vooral van het vwo) die minder last hebben van misconcepties of de geconstateerde misconceptie zelf kunnen oplossen veelal meer komen tot diep leren. Leerlingen die regelmatig tegen misconcepties aanlopen komen veel minder of helemaal niet tot diepleren bij het programmeren.

[2] Binnen dit onderzoek wordt de volgende definitie gehanteerd: een denkbeeld dat een leerling heeft met betrekking tot een programmacomponent die niet overeenkomt met hoe deze geïnterpreteerd wordt of gaat worden door de compiler of parser.

Inleiding

In februari 2016 heeft Stichting Leerplan Ontwikkeling (SLO) een adviesrapport uitgebracht dat het vorige, uit 1998 stammende, examenprogramma informatica moest vervangen. Dit advies resulteerde in een nieuw examenprogramma informatica dat per 1 augustus 2019 is ingegaan. Het kernprogramma van dit nieuwe examenprogramma is opgedeeld in vijf kennisdomeinen namelijk: Grondslagen, Informatie, Programmeren, Architectuur en Interactie.

Aan het kennisdomein Programmeren worden de volgende examendoelen gesteld:

- Programmacomponenten[3] ontwikkelen in een imperatieve[4] programmeertaal;
- Daarbij programmeertaalconstructies gebruiken die abstractie ondersteunen;
- Programmacomponenten zodanig structureren dat ze door anderen gemakkelijk te begrijpen en te evalueren zijn;
- Structuur en werking van gegeven programmacomponenten uitleggen;
- Zulke programmacomponenten aanpassen op basis van evaluatie of veranderde eisen.

Betreffende keuze van een imperatieve programmeertaal laat het SLO over aan de docent. Binnen dit onderzoek is als programmeertaal gekozen voor Python. De reden hiervoor is dat ten tijde van dit onderzoek Python de meest gebruikte programmeertaal in de wereld is (TIOBE Software BV, 2021) (https://www.tiobe.com/tiobe-index/) en Python een veel gebruikte codeertaal is binnen wetenschappelijk onderzoek, met name artificial intelligence, is. Wel stelt het SLO dat binnen het kennisdomein Programmeren de basis-programmeerconstructies (toekenning, opeenvolging, herhaling en keuze, functies (met parameters), testen en debuggen) aan de orde moeten komen.

Aanleiding

Sinds schooljaar 2019-2020 wordt de basis voor programmeren volgens de bovenbeschreven inhoud aangeboden aan de leerlingen informatica in de bovenbouw havo en vwo. Regelmatig gebruiken leerlingen vreemde codeerregels, zoals de Python code in Figuur 1: voorbeeldcode 1. Dit is een voorbeeld uit een programma waarbij gecontroleerd moest worden of de invoer van een gebruiker tussen het getal 0 en 10 ligt:

[3] Omdat software tegenwoordig modulair van opbouw is, wordt niet meer gesproken van een programma maar van programmacomponent.
[4] Imperatief programmeren, ook wel procedureel programmeren genoemd, is een programmeerconcept uit de informatica waarbij programma's opgesteld worden in de vorm van opdrachten die direct uitgevoerd kunnen worden.

```
if getal < 10 or > 0:
```

Figuur 1: voorbeeldcode 1

Na het krijgen van een voor hem "onverklaarbare" foutmelding en het raadplegen van de docent, legde de leerling uit dat hier de code uit Figuur 2 bedoeld was:

```
if getal < 10 or getal > 0:
```

Figuur 2: voorbeeldcode 2

De leerling had zijn/haar gedachtegang "… het getal moet kleiner zijn dan tien of groter zijn dan nul …" letterlijk omgezet in Python programmeercode.
Waarom maakt de leerling deze fout en waarom worden meer van dit soort fouten onafhankelijk van elkaar door verschillende leerlingen gemaakt?
Een andere constatering is als leerlingen, naar hun idee, te veel "onverklaarbare" foutmeldingen krijgen tijdens de parser- of compilatieslag[5] de motivatie om te programmeren en om op zoek te gaan naar extra functionaliteit (denk hierbij aan kleurgebruik, plaatjes, opmaak, verbeterde userinterface, ed.) voor een programmacomponent snel afneemt. In sommige gevallen is de leerling zelfs totaal niet meer gemotiveerd om verder te werken aan de codeer opdracht. Dit wordt bekrachtigd door de onderstaande observatie van een 5 vwo-leerling tijdens een praktijkopdracht programmeren.

> *Een 5 vwo-leerling, schooljaar 2021-2022: "Meneer ik vind het onderdeel programmeren in de lessen het allerleukste onderdeel."*
>
> *Dezelfde 5 vwo-leerling 10 minuten later in dezelfde les: "Meneer, ik neem mijn woorden terug, ik vind er toch niks aan, die kleine fouten die ik maak. Ik ben nu weer een ":" vergeten!"*

Deze storende fouten waar leerlingen tegenaan kunnen lopen tijdens het programmeren worden in de literatuur omschreven als misconcepties. Een misconceptie of verkeerd leerling denkbeeld is een verkeerd beeld van de werkelijkheid in een bepaalde setting (referentie). In de theorie worden

[5] Het vertalen of omzetten wordt compilatie of compileren genoemd. Met compiler wordt voornamelijk een programma bedoeld dat een programma in een hogere programmeertaal vertaalt naar een lagere programmeertaal, meestal assembleertaal of machinecode. De voornaamste reden om broncode te compileren is dan ook het maken van uitvoerbare code. Een parser (van het Engelse to parse, ontleden, en het Latijnse pars, deel) is een computerprogramma, of component van een programma, dat de grammaticale structuur van een invoer volgens een vastgelegde grammatica ontleedt (parset).

misconcepties ook wel alternatieve leerling denkbeelden genoemd, of preconcepties (als wordt benadrukt dat het om intuïtieve beelden gaat, die al voor het onderwijs zijn gevormd), of alternatieve concepties (wanneer zij niet zozeer een verkeerde voorstelling van zaken zijn, die op zich wel logisch is, maar die haaks staat op het wetenschappelijk correcte beeld) (Taconis & Terwel, 1999). Het vermoeden dat misconcepties een met name negatieve invloed hebben op de motivatie is in 2019 onderzocht door Chen, Sonnert, Sadler, Sasselov, & Fredericks (2019). Zij concludeerden dat, net als in de observatie hierboven, de motivatie inderdaad afnam wanneer leerlingen tegen misconcepties aanliepen.

Volgens Taconis en Terwel (1999) ontstaan misconcepties door verschillende oorzaken zoals: intuïtie ideeën en modellen, eigenaardigheden van de taal waardoor een soort 'spraakverwarring ontstaat', een specifieke ervaring (bijvoorbeeld bij krachtwerking), en/of doordat leerlingen moeite hebben om te gaan met abstracte en formele begrippen. Behalve het kennis maken met de stof, moeten voortgezet onderwijs leerlingen ook nog leren wat abstract en formeel gedefinieerde begrippen zijn. Zij hebben de neiging met nauwkeurig gedefinieerde vakbegrippen op een zelfde manier om te gaan als in het dagelijks leven. Zo ontstaat verwarring tussen begrippen als 'gebogen' en 'cirkelvormig', 'variabele' en 'waarde', etc.

In het algemeen kan worden vastgesteld dat misconcepties niet gemakkelijk op te lossen zijn (Tyson, Venville, Harrison, & Treagust, 1997). Om misconcepties te voorkomen is het van dat bij exacte vakken als informatica de leerlingen duidelijk gemaakt wordt dat begrippen formeel en precies zijn en door het versterken van de metacognitieve vaardigheden. Bijvoorbeeld door een goede eenduidige uitleg waarbij de docent actief nagaat of de aangeboden stof ook juist geïnterpreteerd en begrepen wordt. Dit kan bijvoorbeeld met een praktijkopdracht waarin een besproken of behandelde misconceptie is verwerkt door de leerling actief en zelfstandig uitgewerkt wordt. Wanneer de leerling tegen deze (bekende) misconceptie aanloopt is het duidelijk dat de door de docent behandelde stof niet correct is geïnterpreteerd door de betreffende leerling en de misconceptie nog aanwezig is.

Doel

De doelstelling van dit onderzoek is inzicht krijgen in de relatie tussen programmeermisconcepties aan de ene kant en motivatie en diep leren aan de andere kant voor het vak informatica bij leerlingen in het voortgezet onderwijs. De verwachting is dat programmeermisconcepties een negatieve invloed hebben op zowel diep leren als op motivatie.

Als vakdocent informatica en als organisatie is het belangrijk om (programmeer)misconcepties te onderkennen en hier adequaat op in te spelen. Het heeft wel degelijk invloed op niet alleen de motivatie voor het vak informatica maar ook hoe de onderwijs carrière van de leerling verloopt; “Cruciaal voor het verloop van de onderwijs carrière is de wijze waarop in het onderwijs met pre- en misconcepten wordt omgegaan.” (Taconis & Terwel, 1999).

Dit onderzoek is onderverdeeld in drie fasen. De eerste fase is een literatuurstudie naar programmeermisconcepties en diep leren. In de tweede fase wordt een lessenserie uitgevoerd waarin leerlingen de basis van het programmeren aangeleerd krijgen. Hierbij is rekening gehouden met de gevonden misconcepties in fase 1 en worden deze er zoveel mogelijk uitgelicht en besproken. In deze tweede fase zal tijdens de praktijkopdrachten geobserveerd worden welke misconcepties er (nog) bij de leerlingen aanwezig zijn. Tevens zal in de praktijklessen geobserveerd worden of, en in welke mate, leerlingen diep leren toepassen. In fase drie wordt aan de hand van een korte enquête nagegaan of er na de theorielessen en praktijkopdrachten nog programmeermisconcepties aanwezig zijn bij de leerlingen. Samen met de lesobservaties uit fase 2 wordt er een conclusie getrokken in welke mate misconcepties invloed hebben op de motivatie en diepleren van leerlingen.

Theoretisch kader

In dit onderzoek worden een aantal begrippen of termen (misconceptie, diep leren, het vak informatica en lessenserie) gebruikt die hieronder uitgewerkt worden.

Misconceptie

Voor de term misconceptie in de informatica zijn verschillende definities voorhanden. Zo beschrijven Qian & Lehman (2017) misconcepties als “gebrekkige ideeën van studenten, die vaak sterk in strijd zijn met algemeen aanvaarde wetenschappelijke consensus”. Verder beschrijven ze misconcepties als volgt: “misconcepties op zich kunnen waarschijnlijk het beste worden gedefinieerd als fouten in conceptueel begrip”. Een andere beschrijving van misconcepties is “opvattingen van studenten die een systematisch foutenpatroon produceren” (Smith III, diSessa, & Roschelle, 2009). Een andere definitie van een misconceptie die gebruikt wordt is “Een misvatting in de programmeertaal is een bewering die kan worden weerlegd door volledig te redeneren op basis van de syntaxis en/of semantiek van een programmeertaal.” (Chiodini, et al., 2021). Deze laatste definitie sluit aan bij de definitie van een **programmeermisconceptie** die in dit onderzoek wordt gehanteerd: **een denkbeeld dat een leerling heeft met betrekking tot een programmacomponent dat niet overeenkomt met hoe deze geïnterpreteerd wordt of gaat worden door de compiler of parser**.

Diep leren

Diep leren is leren waarbij de leerling nieuwe kennis en vaardigheden bewust integreert in de kennis en vaardigheden die hij/zij al heeft. Om diep te leren voert een leerling zelf cognitieve leeractiviteiten uit zoals, denkactiviteiten die leerlingen in staat stellen om nieuwe informatie te verwerken: nieuwe informatie koppelen aan voorkennis, een begrip in eigen woorden uitleggen, informatie te transfereren naar nieuwe contexten) (Novak, 2002). Met andere woorden diep leren kan omschreven worden als voor de leerling betekenisvol leren.

Informatica is een heel breed vakgebied waarbij de docent zelf invulling kan geven aan de inhoud van het vak waarbij de examendoelen natuurlijk gerespecteerd dienen te worden. Omdat het zo’n breed vakgebied is, is het onmogelijk om leerlingen alles van een kennisdomein te doceren. Daarom wordt er binnen de lessen gestreefd naar diep leren waarbij de leerling zelf zijn kennis voor een bepaald onderwerp uitbreidt op basis van zijn eigen interesse, basiskennis en kunde. Dit bevorderen van het diepleren wordt gedaan via het creëren van een krachtige leeromgeving door:

a) Vakoverstijgend te werken (de stelling van Pythagoras uit de wiskundelessen wordt gekoppeld aan informatica),
b) leerlingen opdrachten te laten uitvoeren die relevant zijn voor de leerlingen (de leerling laten inzien dat bepaalde taken geautomatiseerd kunnen worden),
c) dat de leerling zelf actief kennis construeert (door zelf op zoek te gaan naar uitbreidingen op de opdracht) en
d) het coachen van de leerling (door niet altijd een kant-en-klaar antwoord te geven op een vraag maar een leerling te begeleiden via bijvoorbeeld een website, naar het antwoord en zo de leerling zelf het antwoord laten vinden).

Clephas (2018) beschrijft vier vormen van diepe leeractiviteiten te weten: zelfsturing, relateren en structureren, kritisch verwerken en concreet verwerken. Concreet verwerken omschrijft Clephas als "Koppelen aan eigen ervaringen, praktische betekenis zoeken, probleem oplossen, toepassen, nadenken over toepassingen, concrete voorstelling maken, praktisch voorbeeld geven".

Omdat informatica een enorm breed vakgebied is en het onmogelijk is om alles te doceren is het belangrijk dat leerlingen die verder gaan in het vakgebied van de informatica zelf in staat zijn om op zoek kunnen gaan naar informatie die nodig is om een project of opdracht positief af te ronden. Daarom is er in dit onderzoek besloten om bij de leerlingen diep leren te observeren als concreet verwerken.

Om te komen tot diep leren zal er een solide basis moeten zijn. Zo'n solide basis kan alleen maar opgedaan worden als de leerling geen last heeft van onverklaarbare foutmeldingen, de zogenaamde programmeermisconcepties. Chen, Sonnert, Sadler, Sasselov, & Fredericks (2019) concluderen dat de motivatie afnam als leerlingen tegen misconcepties aanliepen. Kallia & Sentance (2019) merkten al op dat er erg weinig te vinden is in de literatuur met betrekking tot misconcepties en diepleren. Zij concluderen uit eigen onderzoek dat leerlingen die erg veel met misconcepties te maken krijgen niet alleen minder voortgang maken met het leren, maar ook het vertrouwen in hun mogelijkheden om programmeren te gaan leren verliezen (self-efficacy) en een minder hoog niveau behalen met programmeren dan leerlingen die minder gehinderd zijn door misconcepties.

Het vak informatica

Het vak informatica is een heel breed vak met zes verplichte examendomeinen plus minimaal twee extra examendomeinen die gekozen moeten worden uit 12 andere examendomeinen. Dit geeft aan dat de docent in de les weinig tijd heeft voor diepgang. Daarom is in dit onderzoek **diep leren** als volgt gedefinieerd: **de intrinsieke bereidheid en kunde van de leerling om van het kennisdomein**

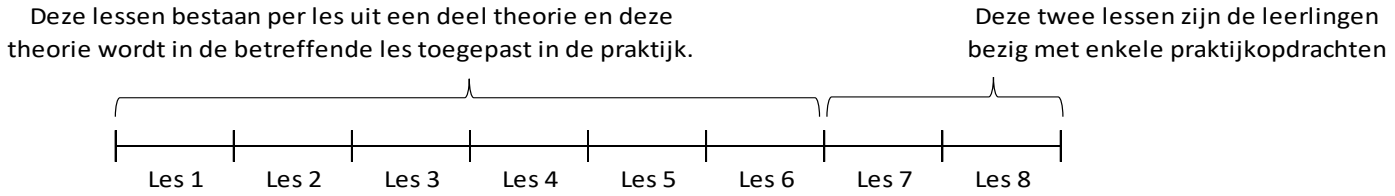
Programmeren zelf extra waardevolle / betekenisvolle kennis op te doen en deze zelfredzaamheid toe te passen in de (praktijk)lessen programmeren.

Lessenserie

In dit onderzoek is aan de hand van een lessenserie bepaald of er tijdens de praktijkdelen van de lessen diep leren is toegepast door de leerlingen. Deze lessenserie bestond uit 8 lessen waarvan de eerste vijf lessen de theorie van het Python programmeren werd behandeld. De meeste theorie is in dezelfde les afgewisseld met het in de praktijk toepassen van de theorie. De rode draad door alle lessen was om uiteindelijk een programma te schrijven welke van een rechthoekige driehoek de lengte van de schuine of lange zijde berekende door gebruik te maken van de stelling van Pythagoras.
Na deze vijf lessen zijn de leerlingen aan de slag gegaan met enkele grotere opdrachten. Enkele voorbeelden hiervan zijn:

- Van twee getallen de som, het verschil, het product en het quotiënt te berekenen waarbij er gecontroleerd moest worden of de getallen tussen 0 en 10 lagen
- Een BubbleSort sorteerroutine schrijven.
- Op een zo efficiënt mogelijke wijze het getal pi tot 35 decimalen achter de komma berekenen met de formule $\pi = 4 * (\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots \frac{1}{n}$.
- Aan de hand van de Turtle bibliotheek cirkels met een random kleur en een random grootte op het scherm laten tekenen.

In Bijlage 1: Lessenserie basis Python programmeren op pagina 50 zijn de dia's van de presentatie van de lessenserie opgenomen. De behandelde onderwerpen zijn beschreven in De lessenserie op pagina 34. Waarbij in de lessen behandelde misconcepties explicatie en nadrukkelijk behandeld en benoemd zijn om zo te voorkomen dat leerlingen tijdens het praktijkprogrammeren gedemotiveerd raakten bij het tegenkomen van een misconceptie.



Figuur 3: Schematischer weergave lessenserie

Onderzoeksvragen

De hoofdvraag van dit onderzoek is: In hoeverre hebben, binnen het kennisdomein Programmeren, programmeermisconcepties invloed op de motivatie en diep leren van leerlingen in het voortgezet onderwijs?

Met de daarbij behorende deelvragen.

1. **Welke misconcepties met betrekking tot (Python) programmeren zijn in de literatuur onderkent?**
2. **Aan welke randvoorwaarden moet voldaan worden opdat leerlingen diep leren gaan toepassen?**
3. **Welke, in de literatuur gevonden, programmeer(mis)concepten kunnen in deze lessenserie "basis Python programmeren" behandeld en besproken worden?**

   Omdat dit onderzoek zich richt op de basis van het Python programmeren zullen niet alle door het SLO gestelde basis-programmeerconstructies behandeld worden omdat bepaalde onderwerpen beter opgenomen kunnen worden in een lessenserie voor gevorderden.

   Op basis van de bovenstaande deelvragen is een lessenserie "basis Python programmeren" ontworpen en is deze lessenserie in de praktijk uitgevoerd.

4. **Welke programmeermisconcepten worden in de praktijklessen waargenomen en welke misconcepties komen naar voren uit de vragenlijst?**
5. **Welke diep leren activiteiten zijn in de praktijklessen programmeren waargenomen?**

Onderzoeksvragen 1 en 2 worden behandeld in fase 1. Onderzoeksvraag 3 vormt fase 2. Waarna in fase 3 (onderzoeksvragen 4 en 5) de data geanalyseerd wordt en op basis hiervan wordt een conclusie gepresenteerd.

Methode

Procedure

Dit onderzoek bestaat uit 3 fasen. Waarbij in fase 1 het literatuuronderzoek plaats vindt naar bestaande en beschreven programmeermisconcepties en er wordt onderzocht wat leerlingen nodig hebben om te komen tot diep leren.
In Fase 2 is aan de hand van het literatuuronderzoek een eerste opzet gemaakt voor een lessenserie, basis Python programmeren, die leerlingen moet stimuleren tot diep leren te komen. In een korte vragenlijst die volgt wordt om de mening van de leerling gevraagd naar aanleiding van de lessenserie maar ook getest of enkele gevonden en goed te toetsen misconcepties nog bestaan. Tijdens de, door de leerlingen, uitgevoerde opdrachten wordt geobserveerd of de eerder gevonden misconcepties nog voorkomen.
In fase drie volgt een analyse van de afgenomen vragenlijst en de lesobservaties. Daarna wordt er een conclusie gepresenteerd met betrekking tot de samenhang tussen programmeermisconcepties aan de ene kant en motivatie en diep leren aan de andere kant.

Opzet lessen

De lessenserie Python programmeren heeft eerste als doel om de leerlingen de basis van het programmeren in Python te leren. Het tweede doel is om de bij onderzoeksvraag 3 gevonden misconcepties tijdens de praktijk te voorkomen door deze misconcepties in de les te benoemen en te behandelen. De rode draad in deze lessenserie is een programma te schrijven die de stelling van Pythagoras oplost. Door het invoeren van twee zijden van een rechthoekige driehoek wordt de derde zijde berekend. Volgens Koopmans (2017) wordt diep leren bevorderd door onder andere vakoverstijgend en samen te laten werken (zie ook Randvoorwaarden diepleren, pagina 30). Deze twee pijlers zijn meegenomen in de opzet van de lessenserie. Bijna iedere les bestaat uit theoretische uitleg met een opzet voor het programma waarna de leerlingen alleen of in tweetallen de code verder uitwerken. Na de theorielessen zijn er 3 lessen gereserveerd om te gaan werken aan opdrachten zoals het maken van een simpele tekst-based rekenmachine, het tekenen van random cirkels met random kleuren in een grafische omgeving en een grafische weergave van de stelling van Pythagoras waar een rechthoekige driehoek getekend moet worden. Bij alle opdrachten moet de leerling minimaal aan de basis voldoen maar creativiteit wordt zeker gewaardeerd. Waarbij creativiteit (er zelf meer van maken dan strikt nodig is) als kenmerk van diep leren wordt opgevat. Het laat immers zien dat de leerlingen zich de kennis en vaardigheid van de oorspronkelijke opdracht zo hebben verworven

en zijn gaan beheersen, dat zij deze verder willen uitbouwen en met nieuwe toepassingen verbinden. Ze zijn dus actief bezig met het construeren van kennis en verwerven van vaardigheden, en willen die bovendien koppelen aan een nieuwe betekenisvolle taak.

Participanten

Het merendeel van de leerlingen die kiezen voor informatica in de bovenbouw, van het havo en vwo, beginnen aan dit vak zonder enige voorkennis. Om een zo representatief mogelijk onderzoek uit te voeren is dit onderzoek afgenomen bij een zo groot mogelijke doelgroep; 4 en 5 havo en 4 en 5 vwo. Deze leerlingen vallen in de leeftijdsgroep van 15 tot 19 jaar waarvan ruim 95% van deze leerlingen jongens zijn.
In totaal hebben 59 leerlingen meegewerkt aan dit onderzoek en hebben deelgenomen aan de lessenserie met de bijbehorende opdrachten. Van de vragenlijst hebben twee leerlingen hebben één vraag opengelaten en heeft één leerling 8 vragen niet beantwoord.
In het begin van het schooljaar is aan alle leerlingen van alle klassen gevraagd of zij onder andere enige ervaring hebben met programmeren in een programmeertaal. Buiten een beetje HTML opmaken van een website in voorgaande jaren tijdens de lessen had geen enkele leerling ooit iets geprogrammeerd. Alle participanten op zowel havo- als vwo-niveau en in alle leerjaren hadden dus aan het begin van dit onderzoek hetzelfde kennisniveau qua programmeren in Python.

Instrumenten

Voor dit onderzoek zijn de volgende instrumenten gebruikt.

Onderzoeksvraag	**Deelnemers**	**Instrumenten**	**Analyse**
Welke misconcepties met betrekking tot (Python) programmeren zijn in de literatuur onderkent?	N.v.t.	Literatuur-studie	Literatuur-studie naar misconcepties binnen het Python programmeren.

Onderzoeksvraag	Deelnemers	Instrumenten	Analyse
Aan welke randvoorwaarden moet voldaan worden opdat leerlingen diep leren gaan toepassen?	N.v.t.	Literatuur-studie	Literatuur-studie naar mogelijkheden om leerlingen diep leren toe te laten passen.
Welke programmeer(mis)concepten kunnen binnen een lessenserie “basis Python programmeren” behandeld en besproken worden?	N.v.t.	Literatuur-studie	Opzet lessenserie Python programmeren.
Welke programmeermisconcepten worden in uitvoering van de lessenserie waargenomen en welke misconcepties komen naar voren uit de vragenlijst?	Leerlingen havo 4,5 en leerlingen vwo 4 en 5.	Vragenlijst en lesobservaties	Analyseren lesobservaties op programmeer-misconcepties.
Welke diep leren activiteiten zijn in de praktijklessen programmeren waargenomen?	Leerlingen havo 4,5 en leerlingen vwo 4 en 5.	Lesobservaties	Analyseren lesobservaties op diep leren.

Vragenlijst

De vragenlijst (zie ook Bijlage 2: Afgenomen vragenlijst, pagina 58) bestaat uit 18 vragen waarvan de eerste 15 vragen een indicatie geven of bepaalde misconcepties nog aanwezig zijn bij de leerlingen. Deze vragen hebben ook betrekking op de technische kant van het Python programmeren. Tien van de eerste 15 vragen zijn meerkeuze vragen. Omdat het in een korte vragenlijst niet mogelijk is om op alle misconcepties te testen zijn alleen de volgende misconcepties in de vragenlijst opgenomen: misconceptie 01, misconceptie 02, misconceptie 05, misconceptie 06, misconceptie 07, misconceptie 08, misconceptie 11 en misconceptie 13. De keuze voor deze beperkte set aan misconcepties is dat misconceptie 01 en misconceptie 02 niet in de praktijk te observeren zijn. Deze misconcepties kunnen alleen bevraagd worden in een theorie- of vragenlijst. Een programmeur zal tijdens het programmeren niet zien hoeveel geheugenruimte er voor een bepaalde variabele gereserveerd gaat worden. Misconceptie 05 is juist opgenomen in de vragenlijst

omdat deze misconceptie voor veel misstanden verantwoordelijk is en erg dicht bij de wiskundelessen ligt en heeft in de informatica een andere definitie heeft en voor leerlingen en zij zien dit nog wel eens als een conflict. Misconcepties 06, 07 en 08 hebben betrekking op loop-constructies en zullen in de praktijk ook niet geobserveerd kunnen worden omdat in deze lessenserie wordt dit aangereikt en dit kan niet vergeten worden. Misconcepties 10 en 11 hebben betrekking op het if-statement waarbij misconceptie 11 een cosmetische misconceptie is die niet conform de "Python naming convention" is geschreven. Ondanks dit zal de code wel correct uitgevoerd worden maar betreft het hier wel "slordig" programmeren. Misconceptie 10 is opgenomen omdat de ervaring leert dat leerlingen het lastig vinden om het concept van boolean in combinatie met een if-statement te bevatten.

Het doel van de vragenlijst is: Kort te testen of de, in de les behandelde, misconcepties door de leerling herkent en opgelost worden. De vragenlijst heeft het karakter van een formatieve toets en is opgesteld op grond van de tien jaar kennis en ervaring als docent wiskunde en informatica. Deze vragenlijst is aangepast na geplaatste opmerkingen van een college docent informatica.

Lesobservaties

De lesobservaties van de misconcepties waar leerlingen tegenaan liepen en de diep leer activiteiten van de leerlingen werden kort genoteerd. Omdat niet alle leerlingen direct ondersteund konden worden, zijn waarschijnlijk enkele misconcepties door de leerling of medeleerling zelf opgelost. Doordat de observaties genoteerd moesten worden en niet alle leerlingen direct ondersteund konden worden bestond er dus een kans dat er minder misconcepties zijn waargenomen dan er werkelijk zijn voorgekomen. Ook bestond de mogelijkheid dat leerlingen die diep leren toepassen in staat zijn geweest om de tegen gekomen misconcepties zelfstandig op te lossen. Deze misconcepties zijn dan ook niet geobserveerd.

De diep leer activiteiten zijn altijd waargenomen in de eindopdrachten van de leerlingen. Als de leerling meer had gedaan dan nodig was, heeft de leerling volgens de definitie van dit onderzoek diep leren toegepast door 'concrete verwerking'. Soms waren leerlingen zo enthousiast over de creativiteit van een klasgenoot dat de leerlingen deze creativiteit ook zelf gingen implementeren. In de observaties is dit uiteraard opgenomen als een diep leer moment.

Analyse

Bij het uitvoeren van de opgezette lessenserie is uit de literatuur bekend welke misconcepties er zich kunnen voordoen. Aangezien de doelgroep bij navraag in het begin van het schooljaar geen enkele

programmeerervaring heeft, is het niet mogelijk om op voorhand na te gaan welke misconcepties er bij deze leerlingen leven. Daarom is tijdens de lessen geobserveerd of er nog misconcepties bestaan. Deze observaties geven een beeld of tijdens of na de theorielessen nog misconcepties en welke misconcepties aanwezig waren. In Bijlage 2: Afgenomen vragenlijst is de vragenlijst zoals deze is afgenomen bij de leerlingen terug te vinden. De betreffende vragenlijst bestaat uit 18 vragen. Met vraag 1 t/m 14 wordt kort getest of bepaalde misconcepties nog aanwezig zijn. De analyse van deze data bestaat uit het bepalen van bij hoeveel procent van de leerlingen na de theorie- en praktijklessen nog misconcepties van een bepaalde soort aanwezig zijn.

De data uit de vragenlijst en de lesobservaties wordt gecombineerd zodat dit een goed beeld geeft van de hoeveelheid en nog aanwezige misconcepties. Na deze analyse is het mogelijk om onderzoeksvraag 6 te beantwoorden en een uitspraak te doen over de relatie tussen programmeermisconcepties en diep leren met die beperking dat waarschijnlijk niet alle misconcepties geobserveerd zijn.

Fase 1

Literatuuronderzoek misconcepties

Zoals beschreven in de Inleiding (pagina 8) heeft het SLO de basis-programmeerconstructies van het kennisdomein Programmeren onderverdeeld in variabelen, toekenning, opeenvolging, herhaling, keuze en functies met en zonder parameters. Voor de indeling van misconcepties binnen het Python programmeren is gekozen voor dezelfde indeling als het SLO voor de basis-programmeerconstructies heeft gedaan, dus in variabelen, toekenning, opeenvolging, herhaling, keuze en functies met en zonder parameters (zie ook Inleiding, pagina 8).

Variabelen

Een variabele binnen een programmeertaal is een stukje geheugen dat tijdens de uitvoer van een programma gereserveerd wordt. Om dit geheugen makkelijk te benaderen wordt er binnen een programmeertaal een naam gegeven aan het gereserveerde geheugen. In dit geheugen kan de programmeur bepaalde waarden zetten. De term variabele is vooral bekend uit de wiskunde, zoals $x = 1$. De waarde is afhankelijk van het soort datatype dat aan de variabele wordt toegewezen. Binnen de programmeertaal Python bestaan de volgende datatypen:

- **int** (integer): een datatype dat alleen gehele getallen kent.
- **str** (string): een datatype dat binnen Python bestaat uit één enkele karakter tot hele stukken tekst.
- **float** (floating point): een datatype dat alle reële getallen omvat. Gehele getallen worden omgezet naar decimale getallen. Dus het getal 5 wordt omgezet naar een 5,0.
- **bool** (boolean): een simpel datatype dat alleen de waarde “True” of “False” kan bevatten.
- **list** (array of lijst): een datatype dat uit meerdere elementen van één of verschillende datatype(n) bestaat.

Leerlingen denken meestal dat voor alle variabelen dezelfde hoeveelheid geheugen gereserveerd wordt (Kaczmarczyk, East, Herman, & Petrick, 2010) (misconceptie 01). Voor het getal pi zal echter meer geheugen gereserveerd worden dan het getal 1 omdat het getal pi uit veel meer individuele cijfers bestaat. Volgens leerlingen wordt er aan variabelen die wel gedeclareerd zijn, maar waaraan nog geen waarde is toegewezen, geen geheugenruimte bezet (Kaczmarczyk, East, Herman, & Petrick, 2010) (misconceptie 02). Ook al heeft een variabele nog geen waarde toch zal het programma ervoor zorgen dat er al geheugen gereserveerd is. Ook zouden, volgens Kaczmarczyk,

East, Petrick, & Herman (2010), leerlingen aannemen dat de index van een lijst begint op 0 en loopt door tot aan de lengte van de lijst (misconceptie 03). De index van een lijst begint op 0 zoals veel tellingen van computers beginnen op 0. Het laatste element van een lijst zit niet op index [lengte van de lijst] maar op [lengte van de lijst – 1] omdat er nu eenmaal geteld wordt vanaf 0 in plaats vanaf 1.

Toekenning

Aan variabelen binnen een programam kan een waarde van een bepaald datatype toegekend worden. In tegenstelling tot in de wiskunde kan de waarde van een variabele gewijzigd worden. Dit gebeurt met behulp van een toewijzing.

Plass (2015) beschrijft dat de misconcepties bij toewijzingen voortkomen uit de wiskunde. Zo is in de wiskunde de vergelijking $x = x + 1$ onoplosbaar. In de informatica echter is dit een toewijzing waarin de variabele x wordt opgehoogd met de waarde één (misconceptie 04). Ook het =-teken is een bron van misconcepties. In de wiskunde is het =-teken een gelijkwaardigheid terwijl in de informatica het =-teken een toekenning betekent (misconceptie 05).

```
if getal = 10:
# in plaats van
if getal == 10:
```

Figuur 4: voorbeeldcode misconceptie 05

Opeenvolging

Met opeenvolging wordt in het SLO-kennisdomein Programmeren bedoeld dat een programma top-down uitgevoerd wordt. Net als bij het lezen van een bladzijde begint de lezer boven aan te lezen. Programmacode wordt ook van boven naar beneden uitgevoerd. Binnen de programmeertaal Python is de opbouw van een programmacomponent ook top-down gericht. Bij leerlingen kan hierover wat verwarring over ontstaan (misconceptie 06). Vooral met de introductie van functies (zie Functies, pagina 25) wil de leerling nog wel eens over het hoofd zien dat een programma top-down uitgevoerd wordt zoals het onderstaande voorbeeld illustreert. Binnen het hoofdprogramma wordt de functie “waardering” aangeroepen en uitgevoerd. Nadat de functie is uitgevoerd wordt keert het programma terug naar het hoofdprogramma en zal de laatste regel, `print("klaar")` ook nog worden uitgevoerd. De uitvoer van het programmavoorbeeld op pagina 22 is dan ook:

```
onvoldoende
Klaar
```

```
# Main
print(waardering(5.4))
print("Klaar")

def waardering(cijfer):
      if cijfer >= 5.5:
            waarde = "voldoende"
      else:
            waarde = "onvoldoende"
      return waarde
```

Figuur 5: voorbeeldcode misconcepties 06

Herhaling

Bij een herhaling (ook wel lus of loop genoemd) wordt een stuk programmacode een aantal keer herhaald. Bijvoorbeeld de getallen 1 t/m 100 op het scherm tonen. Men kan ervoor kiezen om 100 regels code te schrijven die elk een getal op het scherm zetten maar het is handiger om een herhaling in 3 regels code te programmeren; dit maakt de code korter dus overzichtelijker.

Binnen de programmeertaal Python zijn er twee varianten van herhalingen of loops:

- de while-loop en
- de for-loop.

Bij de eerste loop is het van te voren niet bekend hoe vaak de loop doorlopen wordt. Denk hierbij aan dat een programmeur die van te voren niet weet hoevaak men zijn wachtwoord verkeerd invoert. Zolang de gebruiker een verkeerd pincode invoert zal de conditie (uitkomst) van de pincodecontrole "False" zijn.

```
while userPin != bankPin:
      userPin = input("Geef je PINcode in:")
```

Figuur 6: voorbeeldcode while-loop

Bij een for-loop staat het van te voren vast hoe vaak de lus doorlopen wordt. Zolang een teller zijn eindwaarde niet heeft bereikt (conditie is "False") zal de lus herhaald worden.

```
for teller in range(1,10):
      print(teller)
```

Figuur 7: voorbeeldcode for-loop

In het bovenstaande voorbeeld zullen de getallen 1 t/m 10 op het scherm worden getoond.

Zo vonden Sekiya en Yamaguchi (2013) dat herhalingen vaak genegeerd worden door leerlingen (misconceptie 07). In het onderstaande voorbeeld zal eerst de for-loop uitgevoerd worden en zal van

de getallen 0 t/m 9 een voldoende of onvoldoende waardering geven. Daarna zal het programma een waardering geven voor het getal 11. Veel leerlingen negeren de for-loop en geven, als output van de code, alleen de waardering voor het getal 11, "voldoende".

```
def waardering(cijfer):
      if cijfer >= 5.5:
            waarde = "voldoende"
      else:
            waarde = "onvoldoende"
      print(waarde)

# Main
for i in range(10):
      waardering(i)
waardering(11)
```

Figuur 8: Misconceptie 07

Een andere geconstateerde misconceptie is dat er binnen een herhaling gecodeerd wordt met verkeerde variabelen (misconceptie 08).

```
def waardering(cijfer):
      if cijfer >= 5.5:
            waarde = "voldoende"
      else:
            waarde = "onvoldoende"
      print(waarde)

# Main
cijfer = 11
for i in range(10):
      waardering(cijfer)
```

Figuur 9: misconceptie 08

In het bovenstaande voorbeeld wordt binnen de for-loop de functie "waardering" aangeroepen met bij iedere aanroep dezelfde waarde, namelijk cijfer (= 11) in plaats van de waarde van i. De output is dan ook 10 keer "voldoende", de waardering voor het getal 11.

Tevens werd geconstateerd dat de leerling aanduidt dat de programmacode midden in de loop stopt op het moment dat de conditie "False" wordt (misconceptie 09). In Figuur 10 wordt binnen de while-loop de waarde van cijfer gelijk op 11 gezet. Aangezien 11 groter is dan 10 wordt neemt men vaak aan dat de while-loop niet afgemaakt wordt omdat er nu niet meer aan de conditie (cijfer <= 10) voldaan wordt. Na de if-else-statement echter zal de controle van de while-loop opnieuw gecontroleerd worden. Nu pas zal het programma concluderen dat de conditie "False" is geworden en de while-loop beëindigen. De output van de code van Figuur 10 op pagina 25 is "voldoende".

```
# Main
cijfer = 1
while cijfer <= 10:
      cijfer = 11
      if cijfer >= 5.5:
            print(“voldoende”)
      else:
            print(“onvoldoende”)
```

Figuur 10: misconceptie 09

Keuze

Binnen het programmeren in Python is er in de basis maar één vorm van keuze; de if-conditie. Als de conditie van een if “True” is, zal het if-statement uitgevoerd worden. Leerlingen vinden het lastig om een if-conditie correct te beschrijven (Mohamad, Shukor, & Mohtar, 2009). De ervaring leert dat leerlingen niet altijd door hebben dat de conditie van een if-conditie altijd van het datatype boolean moet zijn (misconceptie 10).
Ook wil het wel eens gebeuren dat leerlingen een opeenvolging van if-condities gebruiken in plaats van een if-then-else-constructie (misconceptie 11). Dit laatste zal geen foutmelding opleveren maar het correct gebruiken van een if-conditie kan de leesbaarheid van een programma verbeteren.

```
# Main
zijdeA = 12

if zijdeA <= 12:
      print(“Zijde A is kleiner of gelijk aan 12”)
if zijdeA > 12
      print(“Zijde A is groter dan 12”)

# In plaats van

if zijdeA <= 12:
      print(“Zijde A is kleiner of gelijk aan 12”)
else:
      print(“Zijde A is groter dan 12”)
```

Figuur 11: Voorbeeld verkeerd gebruik if-conditie (misconceptie 11)

Functies

In de informatica is een functie, soms ook subprogramma, subroutine, procedure of routine genoemd, een duidelijk afgebakend programmablok met een eigen naam binnen een computerprogramma. De desbetreffende functie kan elders in het programma aangeroepen (uitgevoerd) worden. Zo is het in het spel boter-kaas-en-eieren mogelijk om een kruisje (X) of rondje (O) op een 9-tal plaatsen te zetten. In plaats van voor iedere plek een aparte code te schrijven voor

het plaatsen van een X of een O, bestaat de mogelijkheid deze code één keer te schrijven in de vorm van een functie. Deze functie om een X of O te plaatsen kan dan overal in het programma aangeroepen worden.
Functies kunnen na aangeroepen te zijn een waarde teruggeven aan het hoofdprogramma. Een functie kan bijvoorbeeld controleren of het ingevoerde wachtwoord correct is. Als het wachtwoord correct is ingevoerd geeft de functie de waarde "True" terug aan het hoofdprogramma. Bij een incorrecte invoer van het wachtwoord geeft de functie de waarde "False" terug aan het programma.

Kallia en Sentance (2019) hebben vonden zeven misconcepties gevonden met betrekking tot het programmeren met functies.
De eerste is dat er in het hoofdprogramma een return statement gegeven dient te worden om de returnwaarde van een functie te kunnen verwerken (misconceptie 12, zie Figuur 12).

```
def waardering(cijfer):
      if cijfer >= 5.5:
            waarde = "voldoende"
      else:
            waarde = "onvoldoende"

# Main
print(waardering(5.4))
return waarde
```

Figuur 12: voorbeeldcode misconceptie 12

Deze code zal een foutmelding genereren tijdens de parser controle. De correcte code is dat de return plaats moet vinden in de functie "`waardering`" aangezien de functie een waarde terug geeft.

Bij het aanroepen van een functie kan een waarde meegegeven worden die binnen de functie verwerkt wordt. In het onderstaande voorbeeld wordt de waarde van "`cijfer`" meegegeven aan de functie "`waardering`".
De tweede gevonden misconceptie is dat leerlingen denken dat de return naam van een functie hetzelfde moet zijn als de functienaam (misconceptie 13). Dit hoeft echter niet en is alleen maar verwarrend. Het is zeer raadzaam om de functienaam en de variabele naam binnen de functie een verschillende naam te geven.

```
def waardering(cijfer):
      if cijfer >= 5.5:
            waardering = "voldoende"
      else:
            waardering = "onvoldoende"
      return waardering

# Main
cijfer = 5.4
print(waardering(cijfer))
```

Figuur 13: voorbeeldcode misconceptie 13

De derde door Kallia en Sentance (2019) gevonden misconceptie heeft betrekking op de meegegeven parameter bij de aanroep van de functie. Hierbij wordt aangenomen dat deze parameter

```
def waardering(cijfer):
      cijfer = 5.4
      if cijfer >= 5.5:
            waarde = "voldoende"
      else:
            waarde = "onvoldoende"
      return waarde

# Main
print(waardering(5.4))
```

Figuur 14: voorbeeldcode misconceptie 14

alsnog binnen de functie een waarde toegewezen moet krijgen (misconceptie 14).
Dit is absoluut niet wenselijk; functies dienen juist flexibel te zijn. Door de regel "`cijfer = 5.4`" in de functie "waardering" weg te laten wordt de functie flexibeler en kan op deze manier van ieder cijfer bepalen of het een on-of voldoende is (zie Figuur 14).

De volgende door Kallia en Sentance (2019) beschreven misconceptie binnen functies is dat de parameter die meegeven wordt in het hoofdprogramma dezelfde parameternaam moet hebben als in de functie (misconceptie 15).

```
def waardering(cijfer):
      if cijfer >= 5.5:
            waarde = "voldoende"
      else:
            waarde = "onvoldoende"
      return waarde

# Main
cijfer = 5.4
print(waardering(cijfer))
```

Figuur 15: voorbeeldcode misconceptie 15

In het bovenstaande voorbeeld wordt de functie “waardering” aangeroepen met de parameter “`cijfer`”. Leerlingen denken vaak dat de parameternaam, in dit voorbeeld “`cijfer`”, ook de naam moet zijn die gebruikt wordt binnen de functie “`waardering`”. Python geeft hier geen foutmelding op maar de variabelenaam binnen de functie mag verschillen van de naam waarmee de functie de parameter aangeroept.

Bij de vijfde misconceptie nemen leerlingen aan dat de return waarde van een functie ook meegenomen moet worden in de parameters bij de aanroep van de functie (misconceptie 16).

```
def waardering(cijfer):
      if cijfer >= 5.5:
            waarde = “voldoende”
      else:
            waarde = “onvoldoende”
      return waarde

# Main
cijfer = input(“Voer een cijfer in:”)
print(waardering(cijfer, waarde))
```

Figuur 16: voorbeeldcode misconceptie 16

Het voorbeeld in figuur 15 illustreert misconceptie 16. De functie “`waardering`” wordt aangeroepen met zowel de waarde die verwerkt moet worden, het “`cijfer`”, als de waarde die door de functie teruggegeven moet worden, de “waarde”. Deze code zal een foutmelding opleveren bij het controleren door de parser omdat de parser verwacht dat de functie “`waardering`” één waarde meekrijgt in plaats van twee waarden.
De correcte code om de functie “`waardering`” aan te roepen is met alleen de parameter cijfer “`print(waardering(cijfer))`”.

De zesde misconceptie met betrekking tot functies is dat leerlingen veronderstellen dat globale variabelen zonder meer gebruikt kunnen worden binnen een functie (misconceptie 17).

```
def opWaardering():
      if cijfer >= 5.5:
            waarde = cijfer + 0.6
      else:
            waarde = cijfer + 0.8
      return waarde

# Main
cijfer = 5.4
print(opWaardering())
```

Figuur 17: voorbeeldcode misconceptie 17

Globale variabelen zijn variabelen die alleen beschikbaar zijn binnen het hoofdprogramma. Deze variabelen zijn dus niet zomaar vanuit een functie te veranderen. Met een kleine aanpassing zijn deze globale variabelen wel oproepbaar binnen een functie in de programmeertaal Python (zie onderstaand voorbeeld).

```
def opWaardering():
      global cijfer
      if cijfer >= 5.5:
            waarde = cijfer + 0.6
      else:
            waarde = cijfer + 0.8
      return waarde

# Main
cijfer = 5.4
print(opwWaardering())
```

Figuur 18:voorbeeldcode correct gebruik globale variabelen

De laatste door Kallia en Sentance (2019) beschreven misconceptie berust erop dat leerlingen denken dat een aangeroepen functie, ook het resultaat op het scherm toont (misconceptie 18). Het hoeft namelijk niet zo te zijn dat een functie daadwerkelijk een resultaat toont op het scherm.

```
def waardering(cijfer):
      if cijfer >= 5.5:
            waarde = "voldoende"
      else:
            waarde = "onvoldoende"
      return waarde

# Main
waardering(5.4)
```

Figuur 19: voorbeeldcode misconceptie 18

In het bovenstaande voorbeeld zal de functie de tekst "`voldoende`" of "`onvoldoende`" teruggeven aan het hoofdprogramma. Het hoofdprogramma krijgt geen print commando om deze tekst af te drukken en zal naar aanleiding van het aanroepen van de functie 'waardering' geen output op het beeldscherm weergeven. Binnen misconceptie 18 wordt blijkbaar gedacht dat indien een functie een return commando heeft dit ook automatisch op het beeldscherm zichtbaar wordt.

In Tabel 1: Opsomming misconcepten per SLO basis-programmeerconstructie zijn de hierboven behandelde programmeermisconcepten afgezet tegen de basis-programmeerconstructies van het SLO.

Misconceptie nr.	Omschrijving	Basis-programmeer-constructie	Pagina
01	Alle variabelen nemen evenveel geheugenruimte in.	Toekenning	21
02	Een variabele waaraan geen waarde is toegekend neemt geen geheugenruimte in.	Toekenning	21

Misconceptie nr.	Omschrijving	Basis-programmeer-constructie	Pagina
03	De index van een lijst loopt tot en met de lengte van de lijst.	Toekenning	22
04	De vergelijking x = x + 1 is onoplosbaar.	Toekenning	22
05	Een =-teken betekend “is gelijk aan”	Toekenning	22
06	De opbouw van een programma is niet altijd top-down.	Opeenvolging	22
07	Herhalingen worden genegeerd.	Herhaling	23
08	Herhaling aanroepen met verkeerde variabele(n).	Herhaling	24
09	Een herhaling wordt direct onderbroken wanneer de conditie “False” wordt binnen de herhaling.	Herhaling	24
10	De if-conditie hoeft niet van het soort boolean te zijn.	Keuze	25
11	Er wordt gebruikt gemaakt van een if-if-constructie in plaats van een if-else-constructie.	Keuze	25
12	Na het aanroepen van een functie, die een waarde retourneert, moet een return statement volgen.	Functies	26
13	Binnen een functie dient de return variabele dezelfde naam te hebben als de functie zelf.	Functies	26
14	De meegeven waarde aan een functie dient binnen de betreffende functie nogmaals toegewezen te	Functies	27
15	De naam van de variabele waarmee een functie aangeroepen wordt, moet dezelfde naam zijn die	Functies	27
16	De return waarde van een functie moet ook meegenomen moet worden in de parameters bij	Functies	28
17	Globale variabelen kunnen zonder meer gebruikt worden binnen functies.	Functies	28
18	Een aangeroepen functie geeft de return waarde automatisch op het scherm.	Functies	29

Tabel 1: Opsomming misconcepten per SLO basis-programmeerconstructie

Het opvalt op dat vooral bij de programmeerconstructie “toekenning” relatief veel programmeermisconcepten kunnen ontstaan. Daarom is het van belang om vooral in de basis relatief veel aandacht te besteden aan het vermijden van misconcepten.

Randvoorwaarden diepleren

Trilling & Fadel (2009) stelden dat leerlingen eerder zullen gaan diepleren als ze worden uitgedaagd en kunnen werken aan onderwerpen waarin zij persoonlijk geïnteresseerd zijn. Volgens Koopman (2017) wordt diepleren bevorderd in een krachtige onderwijsleersituatie die voldoet aan de volgende zes criteria:

- De nadruk van het leren ligt niet alleen op kennis maar ook op de relatie met vaardigheden en houding. Met andere woorden, leer leerlingen niet alleen de programmeertaal maar ook de conventies en afspraken die binnen de Python community gelden en dat een programma van voldoende commentaar voorzien moet zijn.
- Laat de leerlingen vakoverstijgend werken. Bedenk opdrachten die raakvlakken hebben met andere vakken of wellicht iets met interesses van leerlingen.
- Laat de leerlingen werken aan levensechte opdrachten; opdrachten die aansluiten bij het vervolgonderwijs of die van echte opdrachtgevers komen.
- Laat leerlingen samenwerken aan opdrachten en het zelf actief construeren van kennis. Dit laatste kan door middel van diep leren maar ook elkaar in groepsverband van elkaar laten leren.
- De docent richt zich op het coachen of begeleiden van het leerproces van leerlingen en laat leerlingen zelf uitzoeken hoe iets werkt. Als docent stuur je de leerling naar de juiste oplossing.
- De docent geeft de leerling feedback over het product, het proces en de zelfsturing. Dus niet alleen waardering voor het eindresultaat maar ook op extra toevoegingen en het proces van leren.

Fase 2

Behandelde programmeermisconcepties

De voortgezet onderwijs school waar dit onderzoek plaatsvindt heeft het schooljaar verdeeld in 4 blokken van elk ongeveer 10 weken waarvan één week gezien wordt als een toetsweek. Het PTA[6] biedt weinig ruimte om de lessenserie langer te maken. De lessenserie kan dus maximaal 9 weken in beslag nemen.

Alle leerlingen die deelnemen aan dit onderzoek hadden geen enkele Python programmeerervaring. Daarom is besloten om niet alle door het SLO gestelde basis-programmeerconstructies te behandelen in de op te zetten lessenserie. De theorie van functies en de variabele soort array's of lijsten zijn vanwege de hoge begripsvorm achterwege gelaten voor deze beginnende programmeurs. De theorie met betrekking tot functies, array's en lijsten kan eventueel in een vervolgcursus Python programmeren opgenomen worden.

Geschikte didactiek voor het aanleren van programmeerconcepten

Om de instructie zo efficiënt en effectief mogelijk te laten verlopen, zijn de onderstaande 10 tips van Brown & Wilson (2018) geïmplementeerd in de lessenserie.

1) **Onthoud als docent dat programmeren of coderen geen aangeboren bekwaamheid is** maar een competentie welke aangeleerd en verbeterd kan worden door oefening. Het is dus een fabel dat sommige leerlingen het programmeren in zich hebben en andere leerlingen dit niet hebben. Iedereen kan dus leren programmeren.
2) **Gebruik "peer instruction"** zoals Eric Mazur van de Harvard University heeft ontwikkeld. Dit houdt in dat de docent na de uitleg van, een deel van, de theorie een goed doordachte meerkeuzevraag (met eventueel al rekening houdend met misconcepties) introduceert bij de leerlingen. Deze vraag wordt in eerste instantie door de leerlingen individueel beantwoord. Daarna gaan de leerlingen in groepjes, van 2 tot 4 leerlingen, overleggen wat het juiste antwoord zou moeten zijn. Hierdoor wordt de leerling "gedwongen" om zijn gedachtegang onder woorden te brengen. Daarna wordt de meerkeuzevraag in groepjes nog een keer beantwoord. Afhankelijk van de gegeven antwoorden kan de docent verder gaan met de theorie of dieper ingaan op de gegeven antwoorden.

[6] PTA staat voor "programma van toetsing en afsluiting". Hierin is onder andere beschreven welke onderwerpen binnen een periode behandeld gaan worden en hoe de toetsing van het betreffende onderwerp gaat plaats vinden.

Door theorie te bespreken, toe te passen en te evalueren wordt 70% van de stof onthouden (Anderson, 2013).

3) **Pas "live" coderen toe** in plaats van alles van tevoren uitgewerkt te hebben op sheets. Live coderen is dat de docent vanuit het niets een programma opbouwt in plaats van al een (deel)programma kant en klaar hebben staan. Dit heeft de volgende voordelen:
    - Het bevordert de interactie tussen docent en leerling.
    - Het bevordert ondoordachte kennisoverdracht. Bijvoorbeeld dat een programma top-down geschreven en uitgevoerd wordt of veel gebruikte afkortingen (bijv. turtle.pu() in plaats van turtle.penup()).
    - Het remt de docent af zodat de leerlingen bij blijven. De docent kan ook te langzaam gaan waardoor de leerlingen ook afhaken. Zorg als docent dat je weet wat je wilt overbrengen. Wellicht is het ook handig om een kant-en-klaar sjabloon klaar te hebben staan dat eventueel ook gedeeld kan worden met de leerlingen.
    - De leerlingen ervaren dat docenten ook fouten maken en hoe de docent deze fouten oplost. Fouten oplossen wordt bijna nooit uitgelegd maar een beginnende programmeur is hier heel wat tijd mee kwijt.
4) **Laat de leerlingen bedenken wat de code gaat doen.** Programmacode demonsteren heeft niet altijd het gewenste resultaat bij leerlingen. Door leerlingen eerst te laten bedenken wat de code gaat doen, is de leeropbrengst groter.
5) **Programmeer in tweetallen.** Een leerling typt de code, de zg. driver, en de andere leerling levert suggesties en commentaar, de navigator. Enkele keren per lesuur dienen de rollen te worden omgedraaid.
6) **Deel een programma op in kleinere delen.** Enkele leerlingen die kiezen voor informatica in het voortgezet onderwijs hebben programmeerervaring. Deze ervaring wordt meestal opgedaan in de hobbysfeer. De meeste leerlingen zijn dus nog nieuwkomers in het programmeren. Door een programmaopdracht op te delen in kleinere delen en te benoemen wat de bedoeling is van dit deel, kunnen ze dit bij andere opdrachten makkelijker gebruiken. Door opdrachten op te delen in kleinere delen zal de communicatie met de medeleerlingen makkelijker verlopen.
7) **Blijf bij één programmeertaal.** Totdat leerlingen gevorderd zijn is het beter om bij één programmeertaal te blijven. Hierdoor kan niet alleen de syntax van de programmeertaal geleerd worden maar ook de werking van een loop of een if-statement.
8) **Gebruik opdrachten die relevant zijn en aansluiten bij de belevingswereld van de leerling.** Leerlingen zullen de opdrachten hierdoor leuker in plaats van lastig gaan vinden.

9) **Onthoudt dat leerlingen beginners en geen gevorderden zijn.** Ga niet te snel en neem niets als veronderstelde kennis aan.
10) **Leer niet alleen programmeren.** Buiten het coderen moeten beginners ook bedenken hoe een programma opgezet moet worden. Bedenk een opdracht waarbij de code gegeven is maar waarbij de regels code in de juiste volgorde gezet moet worden.

De lessenserie

Het literatuuronderzoek naar misconcepties en diep leren in fase 1 samen met de tips voor een didactiek voor het aanleren van coderen vormen samen de opzet voor de lessenserie. Deze data vormen samen een lijst van ontwerpeisen (OE's) die de randvoorwaarden definiëren waaraan de perfecte lessenserie "basis Python programmeren" zou moeten voldoen. De verschillende soorten datatypen dienen behandeld te worden (zie ook Variabelen, pagina 21).

OE1. Er dient ook aandacht geschonken te worden aan herhalingen, keuze, toekenning, opeenvolging en functies (basis-programmeerconstructies).
OE2. Niet iedere variabele neemt evenveel geheugenruimte in beslag (misconceptie 01, pagina 21)
OE3. Als een variabele gedeclareerd wordt, wordt er voor deze variabele gelijk geheugenruimte gereserveerd (misconceptie 02).
OE4. De index van een lijst begint op 0 en loopt door tot de lengte van de lijst (misconceptie 03, pagina 22).
OE5. Een =-teken is een toewijzing in plaats van een vergelijking (misconceptie 04, pagina 22).
OE6. Een vergelijking in de programmeertaal Python is een =-teken in plaats van een dubbele =-teken (==) (misconceptie 05, pagina 22).
OE7. De ongelijkheidstekens ≤ en ≥ (uit de wiskundeboeken) worden niet herkend maar respectievelijk als <= en >= gebruikt in de programmeertaal Python (misconceptie 06, pagina 22).
OE8. Een programmeertaal wordt vaak verward met een spreektaal (misconceptie 07, pagina 23).
OE9. Een imperatieve programmacode wordt altijd top-down uitgevoerd (misconceptie 08, pagina 24).
OE10. Herhalingen kunnen niet genegeerd worden (misconceptie 09, pagina 24).

OE11. Binnen een herhaling of loop moeten de juiste variabele gebruikt worden (misconceptie 10, pagina 25).

OE12. Een herhaling kan niet midden in de loop stoppen op het moment dat de conditie “” “False” wordt (misconceptie 11, pagina 25).

OE13. De conditie van een if-statement en while-loop is altijd een boolean (misconceptie 12, pagina 26).

OE14. De voorkeur gaat niet uit naar een if-if constructie maar naar een if-else constructie (misconceptie 13, pagina 26).

OE15. In het hoofdprogramma dient geen return statement opgenomen te worden om de return waarde van een functie te verwerken (misconceptie 14, pagina 27).

OE16. De naam van de variabele die binnen een functie geretourneerd wordt, hoeft niet dezelfde naam hebben als de functie zelf (misconceptie 15, pagina 27).

OE17. Als een functie een waarde meekrijgt vanuit het hoofdprogramma is het zinloos om deze waarde te wijzigen (misconceptie 16, pagina 28).

OE18. De naam van de variabele die meegeven wordt aan een functie hoeft niet dezelfde variabele naam zijn binnen die aangeroepen functie (misconceptie 17, pagina 28).

OE19. De variabele die een functie retourneert mag niet met de aan te roepen functie meegegeven worden (misconceptie 18, pagina 29).

OE20. Globale variabelen kunnen binnen een functie wel gelezen maar niet gewijzigd worden (misconceptie 19, pagina 36).

OE21. De geretourneerde waarde van een functie wordt niet zondermeer afgedrukt op het scherm (misconceptie 20, pagina 36).

OE22. Onthoud als docent dat programmeren geen aangeboren bekwaamheid is.

OE23. Pas peer-instructie toe; laat de leerlingen in groepjes overleggen voor een mogelijk antwoord op een vraag.

OE24. Pas “live” programmeren toe. Laat zien hoe een programma vanuit het niets opgebouwd wordt en dat de docent ook fouten kan maken.

OE25. Laat leerlingen beredeneren wat de werking van code kan zijn.

OE26. Laat leerlingen in tweetallen werken; wissel regelmatig af wie het programma typt.

OE27. Deel een programma op in kleinere stukjes code.

OE28. Blijf de complete cursus bij één programmeertaal.

OE29. Gebruik voorbeelden die relevant zijn en de leerlingen aanspreken.

OE30. Leerlingen zijn beginners, ga niet te snel door de stof heen.

OE31. Leer de leerlingen niet alleen de code maar ook hoe men tot een oplossing kan komen.

OE32. Bedenk dat de nadruk van het leren niet alleen ligt op kennis vergaren maar ook op de relatie met vaardigheden en houding.

OE33. Laat leerlingen vakoverstijgend programmeren.

OE34. Laat de leerlingen werken aan levensechte opdrachten; opdrachten die aansluiten bij het vervolgonderwijs of opdrachten die van echte opdrachtgevers komen.

OE35. Laat leerlingen samenwerken aan opdrachten en zelf actief kennis construeren.

OE36. Richt je als docent op het coachen of begeleiden van het leerproces van leerlingen.

OE37. Geef als docent de leerling feedback op het product, het proces en de zelfsturing.

In het meest ideale geval zou een lessenserie ontworpen moeten worden die alle hierboven opgesomde ontwerp eisen (OE) moeten bevatten. De opzet van de lessenserie is echter een basiscursus Python programmeren. De onderwerpen met betrekking tot functies en array's of lijsten zouden hier geen onderdeel van moeten zijn wegens de moeilijkheidsgraad van deze onderwerpen. Daarom komen ontwerpeisen 5, 16 tot en met 22 te vervallen. In Bijlage 1: Lessenserie basis Python programmeren op pagina 50 is de complete lessenserie opgenomen.

In de praktijk geobserveerde programmeermisconcepties

Enkele nog niet in de literatuur gevonden maar wel in de praktijk opgemerkte misconcepties hebben betrekking op de schrijfwijze van een ongelijkheid (misconceptie 19). Hier worden net als in de wiskunde boeken de ongelijkheden "groter dan of gelijk aan" en "kleiner dan of gelijk aan" respectievelijk schreven als ≥ en ≤. De Python compiler kan hier niet mee kan omgaan en zal een foutmelding generen. De correcte schrijfwijze is >= en <=.

Een mogelijke verklaring voor de programmeerconceptie is dat deze ongelijktekens niet goed zijn

```
if getal ≤ 10 or getal ≥ 0:
# of
if getal =< or getal => 10:
# in plaats van
if getal <= 10 or getal >= 0:
```

Figuur 20: voorbeeldcode misconceptie 19

aangeleerd tijdens de reken- of wiskundelessen (Plass-Oude Bos, 2015).

Een ander voorbeeld uit de praktijk van een misconceptie is waarbij de leerling spreek- of schrijftaal verward met programmercode (misconceptie 20). In het onderstaande voorbeeld wil de leerling "als

het getal kleiner is dan 10 en groter is dan 0 is, druk dan de tekst “Tussen 0 en 10” af op het scherm” vertalen naar programmacode. Blijkbaar wordt een natuurlijke taal, zoals onze spreektaal, verwart met een formele en logische taal zoals een programmeertaal.

```
if getal < 10 and > 0:
      print(“Tussen 0 en 10”)

# In plaats van

if getal < 10 and getal > 0:
      print(“Tussen 0 en 10”)
```

Figuur 21: correcte code bij misconceptie 20

Leerlingen vonden het lastig om te bevatten dat een conditie van een if-statement of while-statement een boolean oplevert (zie ook misconceptie 10, pagina 25).

```
getal = 12

if getal:
      print(getal)
```

Figuur 22: misconceptie 21

In het bovenstaande voorbeeld is de conditie van de if-conditie altijd “True”. Er wordt getest op getal en getal is op zich “True” want deze bestaat (misconceptie 21).

Verder viel op dat, ondanks dat hieraan nadrukkelijk aandacht aan besteed is tijdens diverse lessen, de leerlingen vaak vergeten dat een if-statement, een for-loop en een while-loop altijd met een “:” eindigen (zie ook Figuur 2 pagina 9, Figuur 8 pagina 24 en Figuur 10 pagina 25). Aangezien deze manier van schrijven programmeertaal afhankelijk is, is deze valkuil nooit opgenomen als een misconceptie (misconceptie 22). Echter kort na het referen aan de theorie of de leerling laten inzien dat er “iets” vergeten was kwam al snel het “Oja”-moment en verhielp de leerling deze fout.

Ondanks dat functies in Python niet behandeld zijn in de basiscursus waren er een flink aantal leerlingen die zich dit onderdeel eigen hebben gemaakt. Vandaar dat er wel observaties zijn gedaan met betrekking tot functies.

Functies kunnen op elk moment in het programma aangeroepen worden. Sommige functies worden aangeroepen met een waarde. Er zijn ook functies die geen waarde vereisen, zoals in Figuur 17 op pagina 28. Regelmatig werd er geconstateerd dat een functie aangeroepen werd zonder de vereiste haakjes zoals is weergegeven in Figuur 23: misconceptie 2 (misconceptie 23).

```
def waardering():
      if cijfer >= 5.5:
            print("voldoende")
      else:
            print("onvoldoende")

# Main
cijfer = 5.4
waardering
```

Figuur 23: misconceptie 23

De code in bovenstaande voorbeeld geeft geen enkel resultaat terug terwijl er in de functie `waardering` wel een printcommando staat en de functie `waardering` (zonder haakjes) aangeroepen wordt. Wordt deze functie met haakjes aangeroepen (`waardering()`) dan zal het gewenste resultaat wel op het scherm getoond worden.

Veel leerlingen van zowel de havo als het vwo vonden het lastig om met condities om te gaan die in een if-statement of in een while-statement voorkomen. Met name een conditie verzinnen die als antwoord "True" of "False" oplevert was voor veel leerlingen lastig op te zetten. Een verklaring hiervoor zou kunnen zijn dat een programmeertaal een logische taal is en geen spreektaal (zie ook misconceptie 20, pagina 36).

In de praktijk geobserveerd diep leren

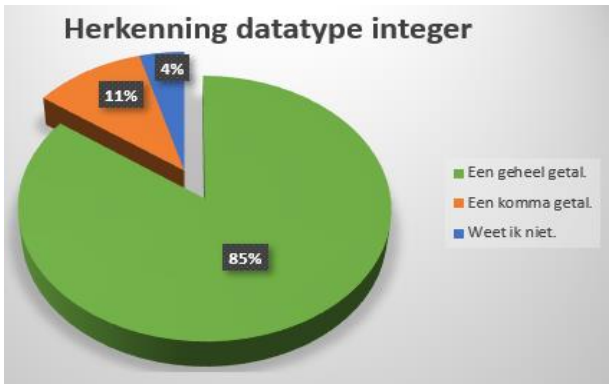
Het enthousiasme van de leerlingen in de lessen was erg hoog. Enkele leerlingen hebben zelfs de opdrachten voor zichzelf uitgebreid. Zo is er een opdracht waarbij een programma geschreven moet worden waarbij 5 verschillend gekleurde schildpadden een traject moeten afleggen. De eerste over de finish is de winnaar. Enkele leerlingen hebben een gok element ingebouwd waarbij men van tevoren kan aangeven welke kleur schildpad de winnaar zal worden. Een ander voorbeeld is een opdracht waarbij de stelling van Pythagoras geïmplementeerd moet worden, inclusief een getekende rechthoekige driehoek. Een vwo leerling vond het wel een uitdaging om de gevraagde maten van twee zijden van deze driehoek in de juiste verhouding op het scherm te tekenen. Enkele leerlingen hebben deze opdracht weten uit te bereiden met een popup-scherm waarin de gebruiker de afmetingen van de diverse zijden van de rechthoekige driehoek kon invullen.

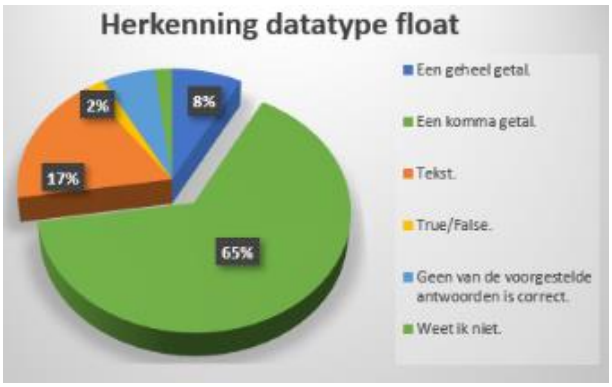
Resultaten vragenlijst

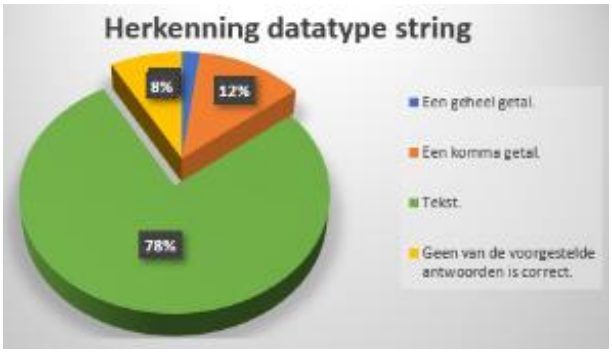
Na het verzorgen van de lessenserie Python programmeren werd bij iedere deelnemer anoniem een vragenlijst afgenomen om te bepalen of de ontwerpeisen voor die leerling behaald zijn (Bijlage 2: Afgenomen vragenlijst, pagina 58). Op deze manier zorgt de vragenlijst ook voor feedback voor mogelijke aanpassingen aan de lessenserie.

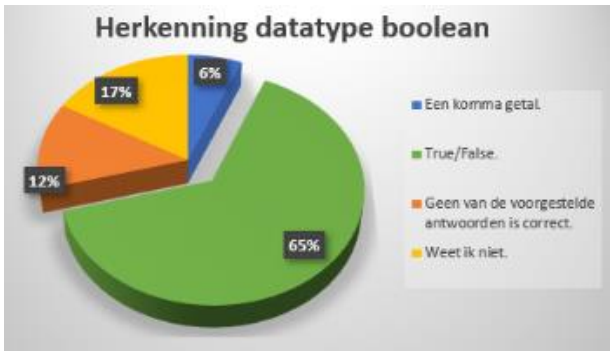
Gezien de gestelde ontwerpeisen aan de lessenserie is het niet mogelijk om via een vragenlijst alle ontwerpeisen te bevragen. De vragenlijst is beperkt tot een aantal ontwerpeisen. In Bijlage 2: Afgenomen vragenlijst is de afgenomen vragenlijst opgenomen en waar mogelijk is het juiste antwoord op de vraag in groen weergegeven.

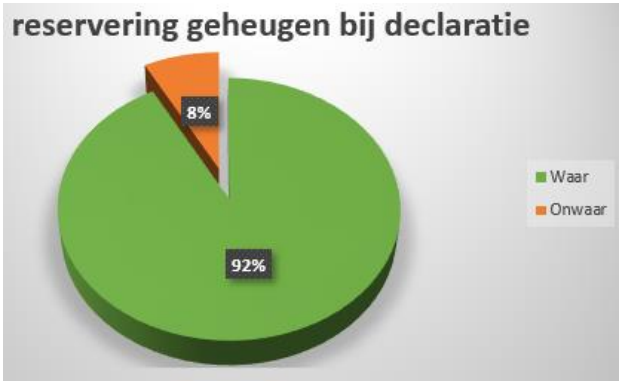
Bij het herkennen van een integer (vraag 1), een float (vraag 2), een string (vraag 3) en een boolean (vraag 4) datatype heeft respectievelijk 85%, 65%, 78% en 65% het juiste antwoord gegeven.



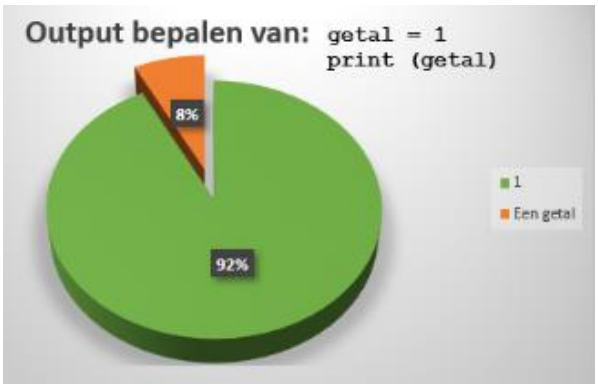






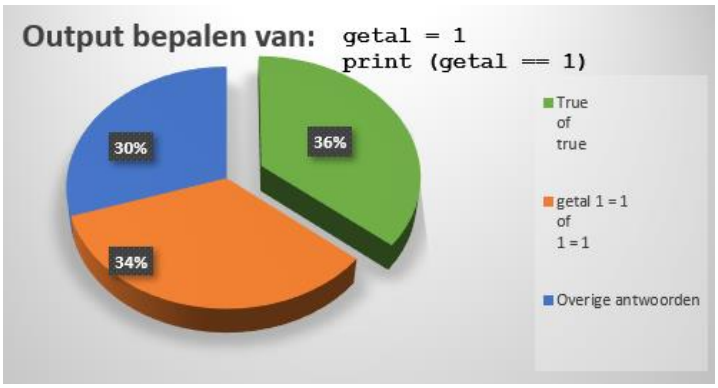


Ruim 90% van de ondervraagde leerlingen wist dat als er een variabele in de programmeertaal Python aangemaakt wordt, hiermee ook gelijk geheugen wordt gereserveerd (vraag 5).

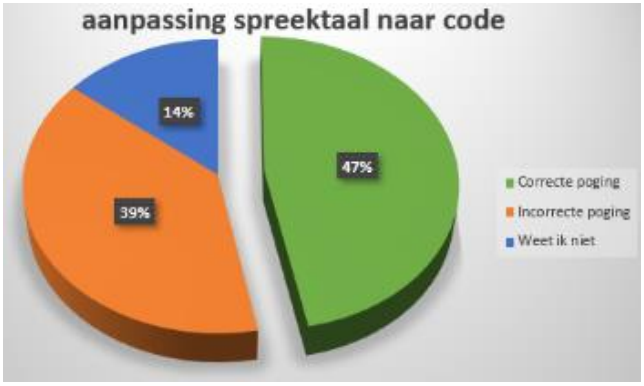


Bij vraag 6 is getest of de leerling de werking van een bepaalde code kan nagaan en er werd gekeken of een leerling wist dat een enkele =-teken (=) een toekenning betekent in plaats van een vergelijking zoals in de wiskunde. Uit de antwoorden op deze openvraag bleek dat ruim 90% van de gevraagde leerlingen zowel de werking van de code konden voorspellen (het getal 1 wordt op het scherm getoond) maar dat men ook wist dat er een toekenning in de code zit.

Bij een soortgelijke open vraag (vraag 7) waarbij buiten het enkele =-teken ook het dubbele =-teken (==)voor kwam, waren het aantal correcte antwoorden een stuk lager. Slechts 36% wist dat de uitvoer van deze code een boolean zou moeten zijn en dus alleen “True” of “False” kon zijn.

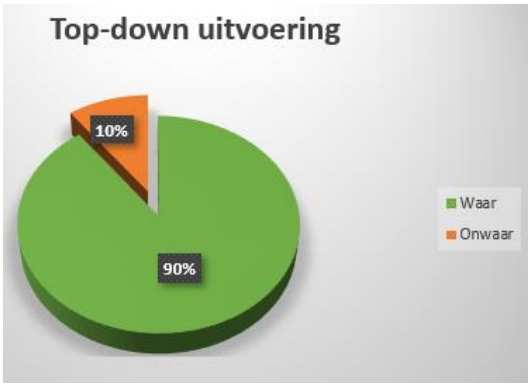


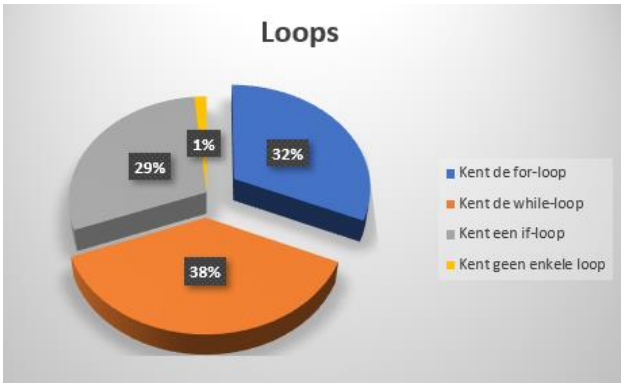
Het standaard toetsenbord bevat niet de tekens “≥” en “≤”. Deze dienen samengesteld te worden met de tekens “<”, “>” “en “=. Slechts 34% van de ondervraagde leerlingen kon bij vraag 8 het teken voor “≤” herschrijven naar “<=”en 37% kon “≥”correct vertalen naar “>=”.



Een computer krijgt zijn commando's van de programmeur via een logische programmeertaal. Python is zo'n logische programmeertaal met opvolgende logische commando's. Het blijkt echter dat net iets minder dan de helft van de ondervraagde leerlingen spreektaal onderscheidt van een programmeertaal. Ongeveer de helft (47%) kon bij vraag 9 de spreektaal herkennen en omschrijven naar logische code.

Negentig procent van de leerlingen wist aan te geven dat een programma top-down uitgevoerd wordt.

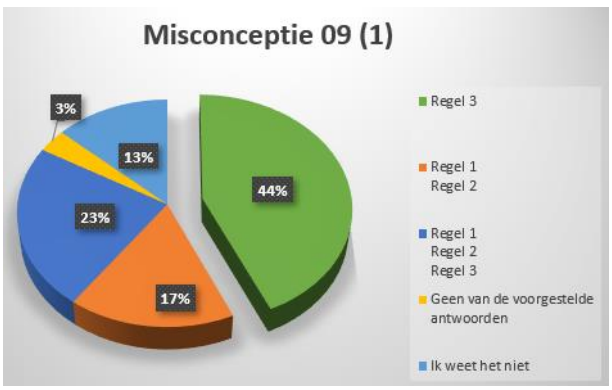




Net als vele andere programmeertalen kent Python verschillende loops of herhalingen (zie ook Herhaling, pagina 23). Deze herhalingen werden in de meeste gevallen ook herkent door de ondervraagde leerlingen. Echter werd een if-statement (keuze) bij 29% van de leerlingen onderkent als een loop. Dit is een, nog niet in de literatuur gevonden, misconceptie.

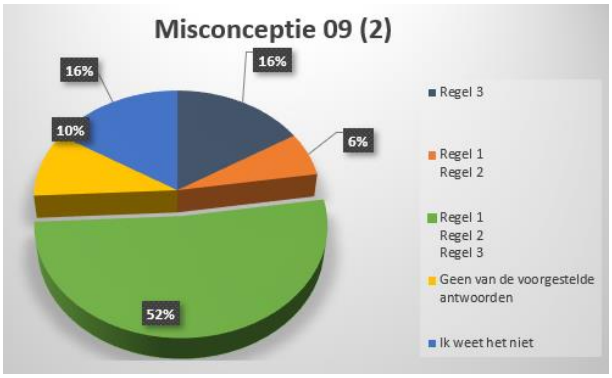
Misconceptie 09 (pagina 24) is op twee verschillende manieren bevraagd. Met de onderstaande code:

```
wachtwoord = "AB"
while wachtwoord != "AB":
    print("Regel 1")
    wachtwoord = "AB"
    print("Regel 2")
print("Regel 3")
```



en

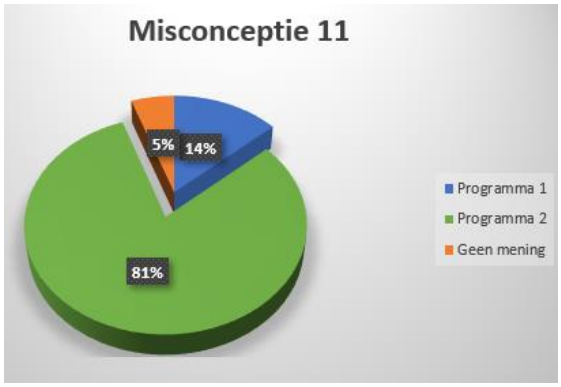
```
wachtwoord = "BA"
while wachtwoord != "AB":
   print("Regel 1")
   wachtwoord = "AB"
   print("Regel 2")
print("Regel 3")
```



Respectievelijk 44% en 52% van de ondervraagde leerlingen wisten het juiste antwoord te geven op de vraag wat de uitvoer van beide programma's was. Een duidelijke indicatie dat misconceptie 09 nog duidelijk aanwezig is.

Misconceptie 11 (pagina 25) betreft het gebruik van de if-statement. Aan de ondervraagden zijn de twee onderstaande programma's voorgelegd met de vraag welke van de twee de voorkeur heeft.

```
# Programma 1:
if wachtwoord == "ABCD":
    print("Wachtwoord is ABCD")
if wachtwoord != "ABCD":
   print("Wachtwoord is geen ABCD")
```




```
# Programma 2:
if wachtwoord == "ABCD":
   print("Wachtwoord is ABCD")
else:
   print("Wachtwoord is geen ABCD")
```

Van de ondervraagden gaf 81% aan een voorkeur te hebben voor variant 2.

Fase 3

Lesobservaties

In de literatuur zijn in totaal 18 misconcepties gevonden die betrekking hebben op (Python) programmeren. Deze misconcepties zijn opgesomd in Tabel 1: Opsomming misconcepten per SLO basis-programmeerconstructie op pagina 30. Na het opzetten en uitvoeren van een basiscursus Python programmeren werden nog enkele, niet in de theorie vermelde misconcepties geobserveerd. Tabel 1 kan daarmee uitgebreid worden met de volgende misconcepties:

Misconceptie nr.	Omschrijving	Basis-programmeer-constructie	Pagina
19	De wiskunde ongelijkheidstekens <= en >= worden verward met ≤ en ≥.	Toekenning en herhaling	36
20	Een programmeertaal wordt vaak geïnterpreteerd als een schrijf- of spreektaal in plaats van een	Toekenning en herhaling	36
21	Een if- of while-conditie aanroepen met alleen een niet-boolean variabele resulteert altijd in	Toekenning, keuze en herhaling	37
22	Een if-, for- en while-statement eindigen altijd met een ":".	Toekenning en herhaling	37
23	Functie zonder parameter aanroepen zonder de verplichte haakjes.	Functies	38

Tabel 2: Niet in de literatuur gevonden misconcepties

In Tabel 3: Geobserveerde misconcepties is per geobserveerde misconceptie het aantal keer dat deze is waargenomen weergegeven. De niet-geobserveerde misconcepties zijn niet opgenomen in deze tabel.

Misconceptie nr.	Omschrijving	Aantal keer en waargenomen
02	Een variabele waaraan geen waarde is toegekend neemt geen geheugenruimte in.	4 keer in vragenlijst
05	Een =-teken betekent "is gelijk aan"	6 keer in vragenlijst 2 keer in praktijk
06	De opbouw van een programma is niet altijd top-down.	6 keer in vragenlijst
10	De if-conditie hoeft niet van het soort boolean te zijn.	27 keer in vragenlijst
11	Er wordt gebruikt gemaakt van een if-if-constructie in plaats van een if-else-constructie.	1 keer in de praktijk
19	De wiskunde ongelijkheidstekens <= en >= worden verward met ≤ en ≥.	16 keer in vragenlijst 3 keer in de praktijk
20	Een programmeertaal wordt vaak geïnterpreteerd als een schrijf- of spreektaal in plaats van een	4 keer in de praktijk

Misconceptie nr.	Omschrijving	Aantal keer en waargenomen
21	Een if- of while-conditie aanroepen met alleen een niet-boolean variabele resulteert altijd in	2 keer in de praktijk
22	Een if-, for- en while-statement eindigen altijd met een ":".	7 keer in de praktijk
23	Functie zonder parameter aanroepen zonder de verplichte haakjes.	5 keer in de praktijk

Tabel 3: Geobserveerde misconcepties

Ondanks dat de meeste misconcepties duidelijk zijn behandeld in de lessen van de basiscursus Python programmeren worden zowel in de lesobservaties als in de antwoorden van de vragenlijst nog steeds misconcepties vastgesteld.

Vooral door de vwo-leerlingen werden producten ingeleverd waarbij de leerlingen creativiteit lieten zien. Bijvoorbeeld bij het programma van de stelling van Pythagoras werden pop-up vensters geïntroduceerd die tijdens de les nooit behandeld zijn. Er was zelfs een 6 vwo-leerling die de rechthoekige driehoek op schaal wilde laten tekenen aan de hand van de twee gegeven lengtes van de zijden van de rechthoekige driehoek. Er kwamen ook toevoegingen naar voren in de vorm van een gok element bij een schildpadden race waarbij de gebruiker kon voorspellen welke schildpad zou winnen. Eén leerling heeft het zelfs voor elkaar gekregen waarbij er daadwerkelijk virtueel geld ingezet kon worden en dit bedrag afhankelijk van winst of verlies respectievelijk verhoogd of verlaagd werd.

Conclusie

Onderzoeksvraag 1: In de literatuur zijn in totaal 18 misconcepties met betrekking tot programmeren gevonden. Deze misconcepties zijn samengevat en opgedeeld in door het SLO gestelde basis-programmeerconstructie in Tabel 1: Opsomming misconcepten per SLO basis-programmeerconstructie op pagina 30. Zeven van de 18 gevonden misconcepties hebben betrekking op functies. Het opvalt op dat vooral bij de programmeerconstructie “toekenning” relatief veel programmeermisconcepten kunnen ontstaan.

Onderzoeksvraag 2: Er is literatuur gevonden waaruit blijkt dat om voor leerlingen te komen tot diep leren er een krachtige onderwijsleersituatie moet zijn waarbij:

- Er niet alleen de nadruk van het leren niet alleen ligt op kennis maar ook op de relatie met vaardigheden en houding van de leerling.
- Leerlingen vakoverstijgend kunnen werken.
- Er gewerkt kan worden aan levensechte opdrachten.
- Leerlingen kunnen samenwerken aan opdrachten en actief zelf kennis construeren.
- De docent een coachende of begeleidende rol aanneemt en feedback geeft op het product, het proces en de zelfsturing van de leerling.

Onderzoeksvraag 3: Omdat de participanten binnen dit onderzoek hebben aangegeven geen enkele ervaring te hebben met een imperatieve programmeertaal is lessenserie opgezet als een basiscursus Python programmeren waarin daarom niet alle basis-programmeerconstructies en de bijbehorende misconcepties behandeld konden worden. Van de programmeerconstructie variabelen is het datatype list niet behandeld en de programmeerconstructie functies is in z’n geheel weggelaten.

Onderzoeksvraag 4: Tijdens de observaties in de lessen zijn vijf misconcepties waargenomen waarvan in de gevonden literatuur geen melding wordt gemaakt, te weten:

Misconceptie nr.	Omschrijving	Basis-programmeer-constructie	Aantal keer geobserveerd
19	De wiskunde ongelijkheidstekens <= en >= worden verward met ≤ en ≥.	Toekenning en herhaling	3
20	Een programmeertaal wordt vaak geïnterpreteerd als een schrijf- of spreektaal in plaats van een	Toekenning en herhaling	4
21	Een if- of while-conditie aanroepen met alleen een niet-boolean variabele resulteert altijd in	Toekenning, keuze en herhaling	2
22	Een if-, for- en while-statement eindigen altijd met een “:”.	Toekenning en herhaling	7

Misconceptie nr.	Omschrijving	Basis-programmeer-constructie	Aantal keer geobserveerd
23	Functie zonder parameter aanroepen zonder de verplichte haakjes.	Functies	5

Tijdens de lessen zijn door observaties de volgende misconcepties, die ook in de literatuur beschreven staan, waargenomen:

- Misconceptie 05: een =-teken betekent “is gelijk aan”, 2 keer waargenomen.
- Misconceptie 11: een dubbele if statement in plaats van een if-else statement, 1 keer waargenomen.

Er is een significant verschil tussen het aantal geobserveerde misconcepties die bekend zijn in de literatuur (3) en het aantal geobserveerde misconcepties die niet in de literatuur gevonden is (21). Een mogelijke verklaring hiervoor is dat de in de literatuur gevonden misconcepties in de lessenserie zijn opgenomen en nadrukkelijk in de lessenserie behandeld zijn. De leerlingen zijn dus redelijk doordrongen van deze misconcepties. Ook zijn de vijf misconcepties die niet in de literatuur gevonden zijn überhaupt niet in de lessen behandeld.

Tussen het aantal geconstateerde misconcepties uit de vragenlijst (60) en de geobserveerde misconcepties uit de praktijklessen zit een behoorlijk verschil (zie ook pagina 44). Dit kan verklaard worden door dat in de vragenlijst een kant en klaar programma is gegeven dat niet door de leerling zelf is ontworpen. Op deze manier kan het dus zijn dat de leerling de ingebouwde misconceptie heeft gemist. Bovendien is de vragenlijst theoretisch en kan de code uit de vragenlijst niet in de praktijk gecompileerd worden waardoor eventuele misconcepties wellicht niet opgemerkt worden door de leerling.

Programma’s kunnen op verschillende manieren gecodeerd worden. Als er tegen een misconceptie aangelopen wordt kan deze in de praktijk wellicht omzeilt worden door op een andere manier te coderen.

Uit tabel 3 blijkt dat er in de praktijk en uit de vragenlijst 10 soorten misconcepties zijn waargenomen. In de praktijk is er in totaal 24 keer een misconceptie geobserveerd. Uit de vragenlijst zijn in totaal 59 misconcepties af te lezen. Dit leidt tot het vermoeden dat de specifieke vorm misconcepties sterk afhangen van onderwijssituatie en leerlingen.

Onderzoeksvraag 5: Ook is tijdens en na de praktijkopdrachten is bekeken of de vorm van het stellen van creativiteit en nieuwe taken van diep leren door de leerlingen zijn toegepast. Met name bijj vwo-leerlingen werd diep leren geobserveerd in de vorm van creatieve vrijheid of aanvullingen van de opdrachten, zoals beschreven is op pagina 43. Waargenomen is dat leerlingen die sneller bekwaam werden in het programmeren dan de gemiddelde snelheid van de klas, ook gemiddeld sneller diep

leren toepasten in de opdrachten. Leerlingen die vaker tegen misconcepties aanliepen kwamen niet tot veel minder toe aan diep leren en ronden de opdrachten af zoals omschreven zonder enige toevoeging of creativiteit vanuit henzelf.

Het geen overziend zou men kunnen concluderen dat wanneer leerlingen tegen veel misconcepties aanlopen dit het diep leren hindert. Dit wordt ondersteunt door (Kallia & Sentance, 2019) die uit eigen onderzoek concluderen dat leerlingen die erg veel met misconcepties te maken krijgen een minder hoog niveau behalen met programmeren dan leerlingen die minder gehinderd zijn door misconcepties.

Aan de andere kant zouden misconcepties een onderdeel kunnen zijn om een programmeertaal aan te leren, door vallen en opstaan. Koopmans (2017) beschrijft dat in een krachtige leeromgeving de nadruk niet alleen op kennis moet zijn maar ook op houding. Het is dus van belang om leerlingen mee te geven dat er fouten gemaakt mogen worden.

Een voorzichtige conclusie zou kunnen zijn dat misconcepties deel uit maken van het leerproces om een programmeertaal te beheersen. Maar als de leerling eenmaal weet welke misconcepties er zijn, de leerling veel minder geneigd is om vast te lopen met misconcepties wat weer leidt tot meer diep leren.

Discussie

In de praktijk is het lastig om zowel te observeren, leerlingen te helpen die vastlopen en goed klassenmanagement uit te voeren. Wellicht dat er meerdere en betere observaties mogelijk geweest als er tijdens de praktijklessen een extra observant aanwezig was geweest. Dit geeft de docent de mogelijkheid om alle leerlingen met vragen en problemen te woord te staan en zo meer misconcepties te observeren. Om een misconceptie door een observant te laten observeren betekent wel dat de observant bekent moet zijn met de misconcepties en de programmeertaal Python. Met andere woorden zou de observant een andere docent informatica of een student van dezelfde opleiding (1ste graad docent informatica) moeten zijn. Of elke les één of twee random leerlingen langere tijd observeren.

Aanbeveling

Relatief zijn de misconcepties die niet vernoemd worden in de literatuur vaker geconstateerd dan misconcepties die wel in de literatuur bekend zijn. De vraag is of als leerlingen tijdens de lessenserie van de niet behandelde misconcepties (misconceptie 19 t/m misconceptie 23) tijdens de les wel akte hadden genomen minder vaak tegen deze misconcepties aangelopen waren. Een aanpassing met betrekking tot de lessen zou een opdracht kunnen zijn die de leerlingen moeten corrigeren, OF een les besteden aan welke acties er genomen kunnen worden als de leerling tegen een foutmelding aanloopt.

Koopmans (2017) stelt dat in een krachtige onderwijsleersituaties er meer diep leren plaats gaat vinden. Wellicht dat een opdracht opgenomen kan worden welke nog meer realistisch of aansprekend is. Of dat de eindopdracht uit één grote opdracht bestaat die door een grote groep leerlingen uitgevoerd wordt. Hiermee wordt het samenwerken nog belangrijker.

Uit de lesobservaties blijkt dat leerlingen erg veel moeite hebben met het concept van boolean in een if-statement of een while-loop. Hier verdient de aanbeveling om dit onderwerp nog meer aandacht te geven. Hetzelfde geldt voor misconceptie 19 waarbij de vakken wiskunde en informatica elkaar, in de ogen van de leerling, tegenspreken.

Samenvattend betekent dit voor de verzorgde lessenserie wordt aanbevolen om het principe van booleans uitgebreider te behandelen. Vooral in combinatie met if-statements of while-loops. Bij if-statements dient men stil te staan die uit meerdere delen bestaan. Denk hierbij aan:

```
if antwoord == "j" or antwoord =="J":
```

die leerlingen vaak noteren als:

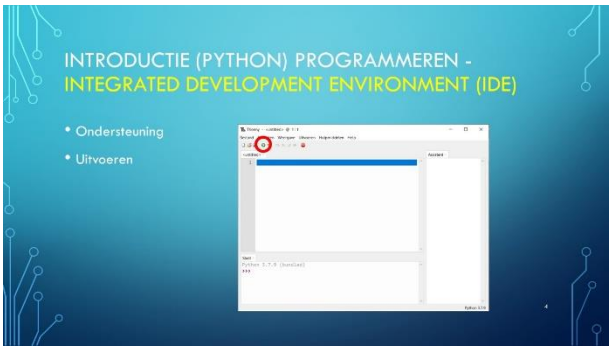
```
if antwoord == "j" or "J":
```

De misconcepties 19 $^{t}/_{m}$ 23 gaan behandelen in de lessen. Deze zijn niet in het literatuuronderzoek gevonden maar wel geobserveerd in de lessen. Deze misconcepties zijn veelvuldig geobserveerd tijdens de lessen.

Tijdens de lessen is er door de leerlingen veel samengewerkt. Dit is door de leerlingen als prettig ervaren en Koopmans (2017) adviseert dit ook om het diep leren te bevorderen.

Bijlage 1: Lessenserie basis Python programmeren

Les 1



INTRODUCTIE (PYTHON) PROGRAMMEREN - VARIABELEN

- Namen
  - Conventie / afspraken
- Soorten
  - integer : heel getal
  - float : komma getal
  - boolean : True / False
  - string : tekst
  - list : meerdere elementen
- Declaratie en toekenning

```
zijdeA = 12  # De eerste rechthoekzijde
```

9

INTRODUCTIE (PYTHON) PROGRAMMEREN - VARIABELEN

- Pas de code van "SvP v0.py" zo aan dat:
  - Er twee variabelen aangemaakt worden: zijdeA en zijdeB
  - De variabele zijdeA krijgt de waarde 12 en
  - de variabele zijdeB een waarde 5 krijgt.
  - Sla dit op als "SvP v1.py"

10

INTRODUCTIE (PYTHON) PROGRAMMEREN - COMMENTAAR

```
# Driver: R. Verheijen
# Navigator: R. Verheijen
'''
Aan de hand van een rechthoekzijde A en rechthoekzijde B wordt
Via de Stelling van Pythagoras de lengte van
de schuine zijde C berekend.
'''
# VARIABELEN
zijdeA = 12 # De eerste rechthoekzijde
zijdeB = 5   # De tweede rechthoekzijde
```

11

INTRODUCTIE (PYTHON) PROGRAMMEREN - VARIABELEN

- Toekenning van andere typen variabelen

```
# VARIABELEN
zijdeA = 12          # Toekenning van een integer
zijdeA = 12 + 5      # Wat zal de waarde van zijdeA zijn?
zijdeB = 5.0         # Toekenning van een float
voorNaam = "Rick"    # Toekenning van een string
bovenDe18 = True     # Toekenning van een boolean
```

12

INTRODUCTIE (PYTHON) PROGRAMMEREN - VARIABELEN

- Maar ook

```
# VARIABELEN
zijdeA = zijdeB = 12          # Welke waarden hebben zijdeA en zijdeB?
# MAIN
zijdeC = zijdeA + zijdeB      # Toewijzing met gelijk een declaratie
                              # Wat is de waarde van zijdeC?
```

13

INTRODUCTIE (PYTHON) PROGRAMMEREN - VARIABELEN

- Pas de code van "SvP v1.py" zo aan dat:
  - Er een variabele programmeurs aangemaakt wordt.
  - Deze variabele krijgt de waarde van zowel de navigator als de driver
  - Sla dit op als "SvP v2.py"

14

INTRODUCTIE (PYTHON) PROGRAMMEREN - COMMENTAAR

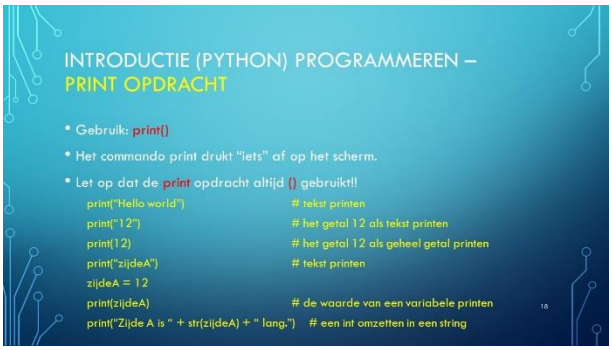
```
# Driver: R. Verheijen
# Navigator: R. Verheijen
'''
Aan de hand van een rechthoekzijde A en rechthoekzijde B wordt
Via de Stelling van Pythagoras de lengte van
de schuine zijde C berekend.
'''
# VARIABELEN
zijdeA = 12 # Eerste rechthoekzijde
zijdeB = 5   # Tweede rechthoekzijde
programmeurs = "R. Verheijen en R. Verheijen"
```

15

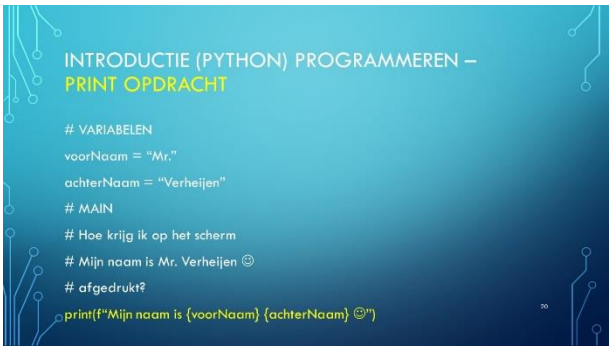
Les 2


LES 2 –
ONDERWERPEN
• Herhaling les 1
• Commentaar gebruik
• Variabelen en toekenning
• Het commando print()
• Vergelijkingen en ongelijkheden
16

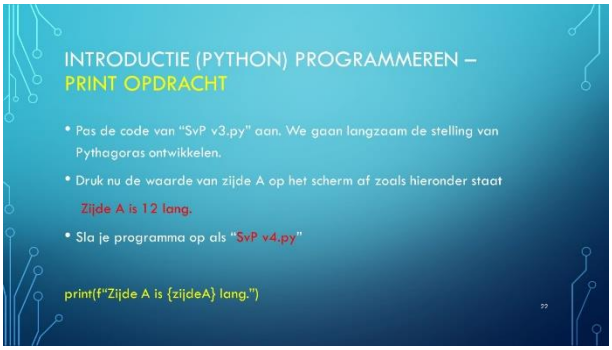

INTRODUCTIE (PYTHON) PROGRAMMEREN -
HERHALING LES 1
Driver: R. Verheijen
Navigator: R. Verheijen
"
Aan de hand van een rechthoekzijde A en rechthoekzijde B wordt
Via de Stelling van Pythagoras de lengte van
de schuine zijde C berekend.
"
# VARIABELEN
zijdeA = 12 # Eerste rechthoekzijde
zijdeB = 5 # Tweede rechthoekzijde
programmeurs = "R. Verheijen en R. Verheijen"
17

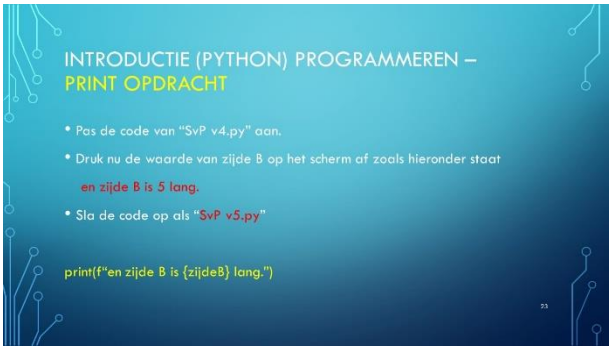

INTRODUCTIE (PYTHON) PROGRAMMEREN –
PRINT OPDRACHT
• Gebruik: print()
• Het commando print drukt "iets" af op het scherm.
• Let op dat de print opdracht altijd () gebruikt!!
print("Hello world") # tekst printen
print("12") # het getal 12 als tekst printen
print(12) # het getal 12 als geheel getal printen
print("zijdeA") # tekst printen
zijdeA = 12
print(zijdeA) # de waarde van een variabele printen
print("Zijde A is " + str(zijdeA) + " lang.") # een int omzetten in een string
18


INTRODUCTIE (PYTHON) PROGRAMMEREN –
PRINT OPDRACHT
• Geavanceerd gebruik met formatting
print(f" ")
# VARIABELEN
zijdeA = 12
# MAIN
print(f"Zijde A is {zijdeA} lang.") # Bedenk in tweetallen wat de output van
deze code is.
19


INTRODUCTIE (PYTHON) PROGRAMMEREN –
PRINT OPDRACHT
# VARIABELEN
voorNaam = "Mr."
achterNaam = "Verheijen"
# MAIN
# Hoe krijg ik op het scherm
# Mijn naam is Mr. Verheijen ☺
# afgedrukt?
print(f"Mijn naam is {voorNaam} {achterNaam} ☺")
20


INTRODUCTIE (PYTHON) PROGRAMMEREN –
PRINT OPDRACHT
• Pas de code van "SvP v2.py" aan. Maak een print opdracht die de namen
van de programmeurs op het scherm afdrukt:
Dit programma is geschreven door: R. Verheijen en R. Verheijen
• Sla je programma op als "SvP v3.py"
print(f"Dit programma is geschreven door: {programmeurs}")
21


INTRODUCTIE (PYTHON) PROGRAMMEREN –
PRINT OPDRACHT
• Pas de code van "SvP v3.py" aan. We gaan langzaam de stelling van
Pythagoras ontwikkelen.
• Druk nu de waarde van zijde A op het scherm af zoals hieronder staat
Zijde A is 12 lang.
• Sla je programma op als "SvP v4.py"
print(f"Zijde A is {zijdeA} lang.")
22


INTRODUCTIE (PYTHON) PROGRAMMEREN –
PRINT OPDRACHT
• Pas de code van "SvP v4.py" aan.
• Druk nu de waarde van zijde B op het scherm af zoals hieronder staat
en zijde B is 5 lang.
• Sla de code op als "SvP v5.py"
print(f"en zijde B is {zijdeB} lang.")
23

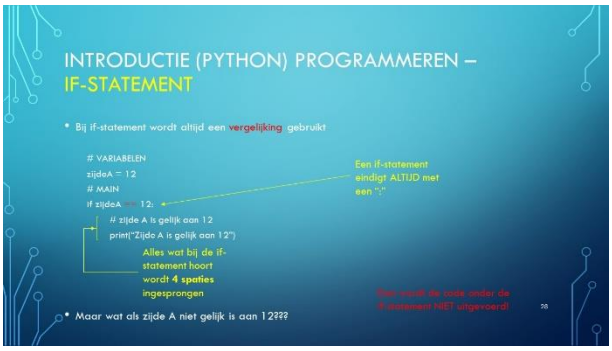

INTRODUCTIE (PYTHON) PROGRAMMEREN - VERGELIJKING
Vergelijking is altijd True / False!!
Wiskundige vergelijkingen
x = 3
x = y
2 = 2
In de programmeertaal Python
# VARIABELEN
zijdeA = 12
zijdeB = 5
# MAIN
print(12 == 13) # Wat zal er op het scherm geprint worden?
print(zijdeA == 12)
print(zijdeB == "5")
24

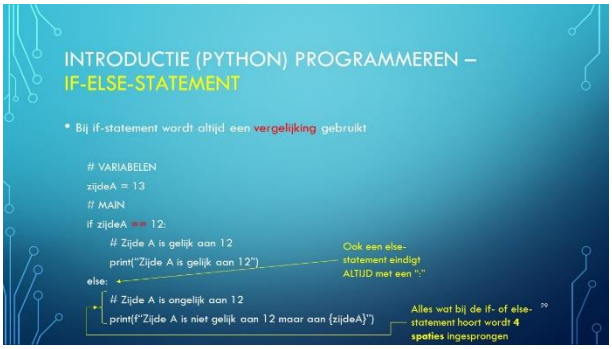

INTRODUCTIE (PYTHON) PROGRAMMEREN - VERGELIJKING
# VARIABELEN
zijdeA = 12
zijdeB = 5
# MAIN
print(zijdeA == zijdeB) #Wat zal de output op het scherm zijn?
print(zijdeA != 12)
print(zijdeB > 12)
print(zijdeB < 12)
print(zijdeA <= 12)
print(zijdeA >= 12)
Bedenk nu zelf een vergelijking en laat je partner de output voorspellen.
25


INTRODUCTIE (PYTHON) PROGRAMMEREN - VERGELIJKING
# VARIABELEN
zijdeA = 12
print(zijdeA == 12)
# Bedenk eerst zelf wat het verschil is tussen de = en de ==.
# Overleg daarna samen over wat het verschil is tussen een = en een ==
26

Les 3


LES 3 –
ONDERWERPEN
• Herhaling les 1 en 2
• Commentaar gebruik
• Variabelen en toekenning
• print()
• Vergelijkingen en ongelijkheden
• Het if-statement
27


INTRODUCTIE (PYTHON) PROGRAMMEREN –
IF-STATEMENT
• Bij if-statement wordt altijd een vergelijking gebruikt
# VARIABELEN
zijdeA = 12
# MAIN
if zijdeA == 12:
# zijde A is gelijk aan 12
print("Zijde A is gelijk aan 12")
Een if-statement eindigt ALTIJD met een ":"
Alles wat bij de if-statement hoort wordt 4 spaties ingesprongen
Hier wordt de code onder de if-statement NIET uitgevoerd!
• Maar wat als zijde A niet gelijk is aan 12???
28

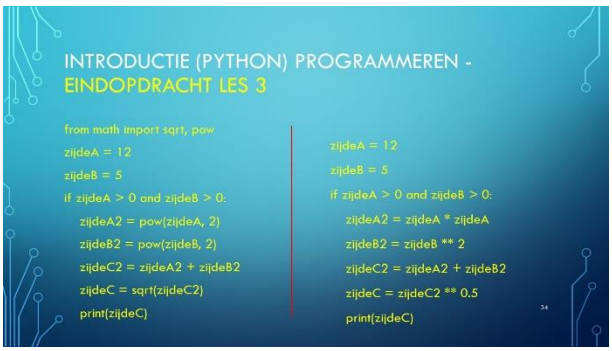

INTRODUCTIE (PYTHON) PROGRAMMEREN –
IF-ELSE-STATEMENT
• Bij if-statement wordt altijd een vergelijking gebruikt
# VARIABELEN
zijdeA = 13
# MAIN
if zijdeA == 12:
# Zijde A is gelijk aan 12
print("Zijde A is gelijk aan 12")
Ook een else-statement eindigt ALTIJD met een ":"
else:
# Zijde A is ongelijk aan 12
print(f"Zijde A is niet gelijk aan 12 maar aan {zijdeA}")
Alles wat bij de if- of else-statement hoort wordt 4 spaties ingesprongen
29


INTRODUCTIE (PYTHON) PROGRAMMEREN –
IF-ELSE-STATEMENT
• Denk in tweetallen wat de output van het volgende programma is
# VARIABELEN
leeftijd = 18
# MAIN
if leeftijd >= 18:
print(f"Ober, een pintje graag! Ik ben namelijk {leeftijd} jaar oud.")
if leeftijd < 18:
print(f"Ober, een cola graag! Ik ben pas {leeftijd} jaar oud.")
30

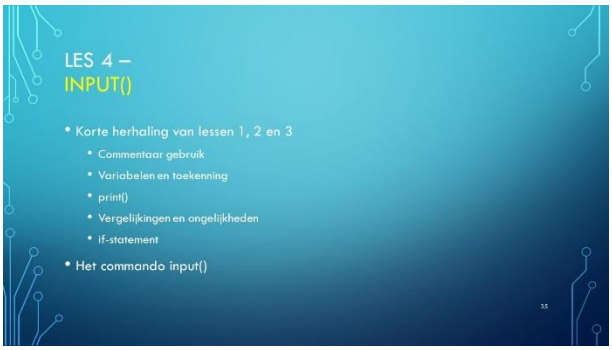

INTRODUCTIE (PYTHON) PROGRAMMEREN –
IF-ELSE-STATEMENT
• Denk in tweetallen wat de output van het volgende programma is
# VARIABELEN
leeftijd = 18
# MAIN
if leeftijd >= 18:
print(f"Ober, een pintje graag! Ik ben namelijk {leeftijd} jaar oud.")
else:
print(f"Ober, een cola graag! Ik ben pas {leeftijd} jaar oud.")
31


INTRODUCTIE (PYTHON) PROGRAMMEREN –
IF-ELSE-STATEMENT
# VARIABELEN
leeftijd = 18
# MAIN
if leeftijd >= 18:
print(f"Ober, een pintje graag")
if leeftijd < 18:
print(f"Ober, een cola graag")
# VARIABELEN
leeftijd = 18
# MAIN
if leeftijd >= 18:
print(f"Ober, een pintje graag")
else:
print(f"Ober, een cola graag")
32

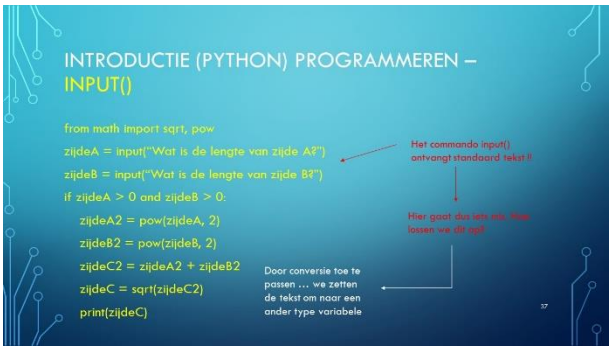

INTRODUCTIE (PYTHON) PROGRAMMEREN -
EINDOPDRACHT LES 3
• Vul je programma "SvP v5.py" aan met een stukje code die het volgende beredeneerd:
• Als zijdeA en zijdeB groter zijn dan 0
• De variabele zijdeA2 gelijk maakt aan zijdeA * zijdeA en
• zijdeB2 gelijk maakt aan zijdeB ** 2
• zijdeC2 gelijk maakt aan zijdeA2 + zijdeB2
• zijdeC uitrekent (weet iemand hoe?)
from math import sqrt
zijdeC = sqrt(c2) of zijdeC = zijdeC2 ** 0.5
• Denk aan je commentaar
• Sla het programma op als "SvP v5.py"
33

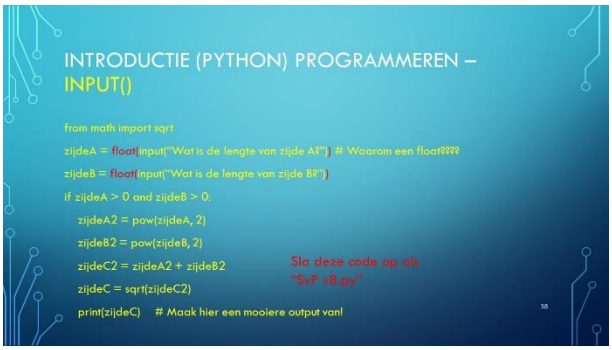

INTRODUCTIE (PYTHON) PROGRAMMEREN -
EINDOPDRACHT LES 3
from math import sqrt, pow
zijdeA = 12
zijdeB = 5
if zijdeA > 0 and zijdeB > 0:
zijdeA2 = pow(zijdeA, 2)
zijdeB2 = pow(zijdeB, 2)
zijdeC2 = zijdeA2 + zijdeB2
zijdeC = sqrt(zijdeC2)
print(zijdeC)
zijdeA = 12
zijdeB = 5
if zijdeA > 0 and zijdeB > 0:
zijdeA2 = zijdeA * zijdeA
zijdeB2 = zijdeB ** 2
zijdeC2 = zijdeA2 + zijdeB2
zijdeC = zijdeC2 ** 0.5
print(zijdeC)
34

Les 4

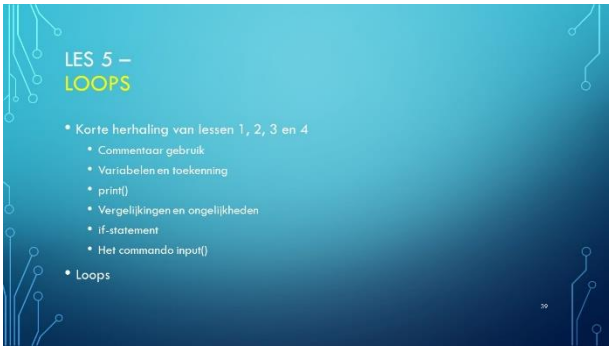

LES 4 –
INPUT()
• Korte herhaling van lessen 1, 2 en 3
• Commentaar gebruik
• Variabelen en toekenning
• print()
• Vergelijkingen en ongelijkheden
• if-statement
• Het commando input()
35


INTRODUCTIE (PYTHON) PROGRAMMEREN –
INPUT()
• Met het commando input() kan een gebruiker tekst invoeren via het toetsenbord
• zijdeA = input()
• zijdeA = input("Hoe lang is zijde A? ")
• Pas het programma "SvP v6.py" zo aan dat het programma om de lengte van zijde A en zijde B vraagt.
• Laat de controle door de if-statement intact en laat het printen van de lengte van zijde C in tact
• Sla het bestand op als "SvP v7.py"
36


INTRODUCTIE (PYTHON) PROGRAMMEREN –
INPUT()
from math import sqrt, pow
zijdeA = input("Wat is de lengte van zijde A?")
zijdeB = input("Wat is de lengte van zijde B?")
if zijdeA > 0 and zijdeB > 0:
zijdeA2 = pow(zijdeA, 2)
zijdeB2 = pow(zijdeB, 2)
zijdeC2 = zijdeA2 + zijdeB2
zijdeC = sqrt(zijdeC2)
print(zijdeC)
Het commando input() antvangt standaard tekst !!
Hier gaat dus iets mis. Hoe lossen we dit op?
Door conversie toe te passen … we zetten de tekst om naar een ander type variabele
37


INTRODUCTIE (PYTHON) PROGRAMMEREN –
INPUT()
from math import sqrt
zijdeA = float(input("Wat is de lengte van zijde A?")) # Waarom een float???
zijdeB = float(input("Wat is de lengte van zijde B?"))
if zijdeA > 0 and zijdeB > 0:
zijdeA2 = pow(zijdeA, 2)
zijdeB2 = pow(zijdeB, 2)
zijdeC2 = zijdeA2 + zijdeB2
zijdeC = sqrt(zijdeC2)
print(zijdeC) # Maak hier een mooiere output van!
Sla deze code op als "SvP v8.py"
38

Les 5

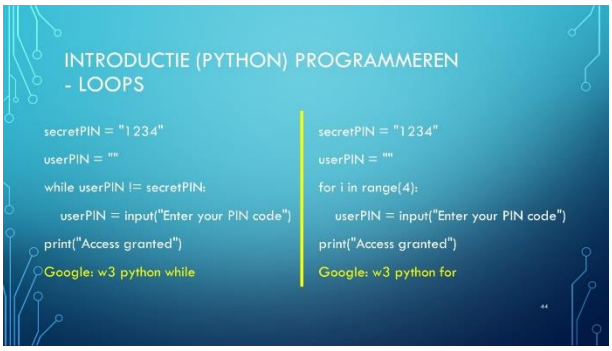

LES 5 –
LOOPS
• Korte herhaling van lessen 1, 2, 3 en 4
• Commentaar gebruik
• Variabelen en toekenning
• print()
• Vergelijkingen en ongelijkheden
• if-statement
• Het commando input()
• Loops
39


INTRODUCTIE (PYTHON) PROGRAMMEREN
- LOOPS
• Loops
• Lus / herhaling
• while => het is van te voren NIET bekend hoe vaak deze loop uitgevoerd wordt
• for => Het is van te voren WEL bekend hoe vaak deze loop uitgevoerd wordt
40


INTRODUCTIE (PYTHON) PROGRAMMEREN –
WHILE LOOP
• Gebruik:
while (conditie):
# Hier komt de code die uitgevoerd wordt zolang conditie True is
zijdeA = 0 # Waarom doen we dit ????
while zijdeA <= 0:
zijdeA = float(input("Geef een correcte lengte voor zijde A, dus groter dan 0: "))
41


INTRODUCTIE (PYTHON) PROGRAMMEREN –
WHILE LOOP
• Bedenk in tweetallen wat de output van de volgende code is.
zijdeA = 0
while zijdeA <= 0:
zijdeA = 12
print(f"Denk je dat deze regel nog op het scherm komt?")
• Probeer het anders uit.
Wees er dus op bedacht dat een loop altijd afgemaakt wordt!!
42


INTRODUCTIE (PYTHON) PROGRAMMEREN
- FOR LOOP
• Gebruik:
for i in range(4):
# Hier komt de code die 4 keer uitgevoerd wordt
• Wat zal de code hieronder doen?
for i in range(4):
zijdeA = float(input("Geef een correcte lengte voor zijde A, dus groter dan 0: "))
43


INTRODUCTIE (PYTHON) PROGRAMMEREN
- LOOPS
secretPIN = "1234"
userPIN = ""
while userPIN != secretPIN:
userPIN = input("Enter your PIN code")
print("Access granted")
Google: w3 python while
secretPIN = "1234"
userPIN = ""
for i in range(4):
userPIN = input("Enter your PIN code")
print("Access granted")
Google: w3 python for
44


INTRODUCTIE (PYTHON) PROGRAMMEREN –
LOOPS
• Schrijf nu een programmaatje dat de getallen 1 t/m 100 afdrukt op het scherm. Doe dit met zowel een for- als een while loop.
'''
Dit programma drukt de getallen 1 t/m 100 af op het scherm.
'''
for i in range(1,101,1):
print(i)
'''
Dit programma drukt de getallen 1 t/m 100 af op het scherm.
'''
i = 1
while i <= 100:
print(i)
i = i + 1 # i += 1
45


INTRODUCTIE (PYTHON) PROGRAMMEREN –
LOOPS
• Pas het programma "SvP v8.py" zo aan dat er net zo lang voor zijde A een lengte gevraagd wordt totdat de lengte > 0
• Doe dit ook voor zijde B
• Zorg voor een mooie output van de lengte van zijde C
• Denk aan je commentaar !!
• Sla je programma op als "SvP v9.py"
46

Les 6, 7 en 8


INTRODUCTIE (PYTHON) PROGRAMMEREN –
LES 6+7: ZELFSTANDIG WERKEN
• In deze les gaan we beginnen met het perfectioneren van ons programma.
• Pas het programma zo aan dat de computer vraagt welke zijde van de driehoek de gebruiker wilt berekenen a, b of c
• Welke stappen moeten er genomen worden als de gebruiker zijde a wilde berekenen? Bedenk dit in tweetallen.
• Pas je programma aan zodat zijde a berekend kan worden.
• Test je programma met de volgende waarden:
• Zijde a = 13
• Zijde c = 5
• Sla je programma op als "SvP v10.py"
47


INTRODUCTIE (PYTHON) PROGRAMMEREN –
LES 6+7: ZELFSTANDIG WERKEN
• Welke stappen moeten er genomen worden als de gebruiker zijde b wilde berekenen? Bedenk dit in tweetallen.
• Pas je programma aan zodat zijde b berekend kan worden.
• Test je programma met de volgende waarden:
• Zijde b = 13
• Zijde c = 5
• Sla je programma op als "SvP v11.py"
48


INTRODUCTIE (PYTHON) PROGRAMMEREN –
LES 6+7: ZELFSTANDIG WERKEN
• Welke stappen moeten er genomen worden als de gebruiker zijde c wilde berekenen? Bedenk dit in tweetallen.
• Pas je programma aan zodat zijde c berekend kan worden.
• Sla je programma op als "SvP v12.py"
49


INTRODUCTIE (PYTHON) PROGRAMMEREN –
VOLGENDE LES (LES 8)
• Een kleine vragenlijst om te checken wat er is onthouden
• En een korte enquête omdat ik jullie mening en input waardeer.
50

Bijlage 2: Afgenomen vragenlijst

Elke leerling die de basiscursus Python programmeren actief heeft gevolgd heeft ook anoniem een vragenlijst ingevuld om inzicht te krijgen in de nog bestaande misconcepties die tijdens de lesobservaties niet gezien zijn. Deze vragenlijst bestond uit de volgende vragen waarbij iedere vraag beschreven staat op welke ontwerpeis deze vraag getoetst wordt. De vragenlijst is in random volgorde afgenomen en de antwoorden werden per vraag in random volgorde gepresenteerd.

Vraag 1: Geef aan van welk datatype de soort variabele integer is.

a) Een geheel getal
b) Een komma getal
c) Tekst
d) True/False
e) Geen van de voorgestelde antwoorden is correct
f) Weet ik niet

Vraag 2: Geef aan van welk datatype de soort variabele float is.

a) Een geheel getal
b) Een komma getal
c) Tekst
d) True/False
e) Geen van de voorgestelde antwoorden is correct
f) Weet ik niet

Vraag 3: Geef aan van welk datatype de soort variabele string is.

a) Een geheel getal
b) Een komma getal
c) Tekst
d) True/False
e) Geen van de voorgestelde antwoorden is correct
f) Weet ik niet

Vraag 4: Geef aan van welk datatype de soort variabele boolean is.

a) Een geheel getal
b) Een komma getal
c) Tekst
d) True/False
e) Geen van de voorgestelde antwoorden is correct
f) Weet ik niet

Vraag 5: Is de volgende stelling waar of onwaar?

Bij het declareren van een variabele wordt al geheugen gereserveerd.

a) Fout
b) Goed

Vraag 6: Wat zal de output van de onderstaande code zijn?

```
getal = 1
print(getal)
```

Vraag 7: Wat zal de output van de onderstaande code zijn?

```
getal = 1
print(getal == 1)
```

Vraag 8: Informatica is een samensmelting van de gebieden elektronica en wiskunde. Uit de wiskunde ken je de tekens ≥ en ≤. In de programmeertaal Python worden deze anders geschreven. Welke bewering(en) is (zijn) waar?

a) ≥ wordt geschreven als >=
b) ≤ wordt geschreven als >=
c) ≤ wordt geschreven als <=
d) ≥ wordt geschreven als <=
e) ≥ wordt geschreven als =>
f) ≥ wordt geschreven als =<
g) ≤ wordt geschreven als =>
h) ≤ wordt geschreven als =<

i) Weet ik niet

Vraag 9: In de onderstaande code wordt er in een if-statement gecontroleerd of zowel de variabele getal groter is dan 12 en kleiner is dan 24. Echter de code klopt niet, geef de correcte code.

```
if getal1 > 12 and < 24:
    # Do something
```

Vraag 10: Python hoort bij de categorie imperatieve programmeertalen. De code van Python wordt altijd top-down uitgevoerd.

a) Fout
b) Goed

Vraag 11: Binnen de programmeertaal Python zijn verschillende herhalingen (loops) mogelijk. Geef aan welke herhalingen bij jou bekend zijn.

a) if - loop
b) do ... while - loop
c) while - loop
d) for - loop
e) Ik weet het niet

Vraag 12: Bekijk de volgende Python code:

```
wachtwoord = "AB"
while wachtwoord != "AB":
   print("Regel 1")
   wachtwoord = "AB"
   print("Regel 2")
print("Regel 3")
```

Wat is de uitvoer van bovenstaande Python code?

a) Regel 1
b) Regel 2
c) Regel 3
d) Regel 1
   Regel 2
e) Regel 2
   Regel 3

f) Regel 1
   Regel 3
g) Regel 1
   Regel 2
   Regel 3
h) Geen van de voorgestelde antwoorden
i) Ik weet het niet

Vraag 13: Bekijk de volgende Python code:

```
wachtwoord = "BA"
while wachtwoord != "AB":
   print("Regel 1")
   wachtwoord = "AB"
   print("Regel 2")
print("Regel 3")
```

Wat is de uitvoer van bovenstaande Python code?

a) Regel 1
b) Regel 2
c) Regel 3
d) Regel 1
   Regel 2
e) Regel 2
   Regel 3
f) Regel 1
   Regel 3
g) Regel 1
   Regel 2
   Regel 3
h) Geen van de voorgestelde antwoorden
i) Ik weet het niet

Vraag 14: Hieronder staan twee Python programma's.

```
# Programma 1:
if wachtwoord == "ABCD":
    print("Wachtwoord is ABCD")
```

```
if wachtwoord != "ABCD":
   print("Wachtwoord is geen ABCD")


# Programma 2:
if wachtwoord == "ABCD":
    print("Wachtwoord is ABCD")
else:
   print("Wachtwoord is geen ABCD")
```

Beschrijf kort welke van de twee programma's jouw voorkeur heeft en leg ook uit waarom je dit vind.

Vraag 15: Voordat je begon aan de lessenserie Python programmeren had je bepaalde kennis met betrekking tot Python programmeren. Zou je kunnen omschrijven welke kennis verandering er heeft plaats gevonden.

Vraag 16: In de lessen zijn verschillende voorbeelden gebruikt om jou een goed beeld te geven van de programmeertaal Python. Welke zijn jou goed bijgebleven?

Vraag 17: Na de lessenserie ben je begonnen met programmeren. Het is onmogelijk om alles wat met Python coderen in een paar lessen samen te vatten. Heb je in de lessen na de lessenserie nog op internet gezocht naar kennis met betrekking tot het Python programmeren?
Zo ja: kun je hiervan een voorbeeld geven.
Zo nee: Kun je aangeven waarom niet.

Vraag 18: Welk cijfer zou je in het algemeen de lessenserie Python programmeren geven?

Referentielijst

Anderson, H. M. (2013, September 4). Dale's Cone of Experience. Opgeroepen op Mei 12, 2021

Barendsen, E., & Tolboom, J. (2016). *Advies examenprogramma informatica havo/vwo.* Enschede: SLO. Opgeroepen op januari 13, 2021

Biemans, H. J. (1997). *Fostering activation of prior knowledge and conceptual change.* Nijmegen: Katholieke Universiteit Nijmegen.

Boschma, R. (2020). *Voorkomen van misconcepties bij introductie programmeren in het voortgezet onderwijs.* Twente.

Brown, N. C., & Wilson, G. (2018). *Ten quick tips for teaching programming.* Opgehaald van PLOS Computational Biology: https://doi.org/10.1371/journal.pcbi.1006023

Chen, C., Sonnert, G., Sadler, P. M., Sasselov, D., & Fredericks, C. (2019, 12 18). The impact of student misconceptions on student persistence in a MOOC. *Journal of Research in Science Teaching*, pp. 879-910. doi:10.1002/tea.21616

Chiodini, L., Santos, I., Gallidabino, A., Tafliovich, A., Santos, A., & Hauswirth, M. (2021). A Curated Inventory of Programming Language Misconceptions. *ITiCSE '21: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (pp. 380–386). New York: Association for Computing Machinery. doi:10.1145/3430665.3456343

Clephas, J. (2018). *Diep leren in Leeratelier Noordoost: Welke vormen van diepe leeractiviteiten en kenmerken van een krachtige onderwijssituatie zijn te herkennen in de ontwerponderzoeken van de deelnemende studenten?* Eindhoven School of Education, Technische Universiteit Eindhoven, Eindhoven. Opgeroepen op 03 12, 2023, van https://pure.tue.nl/ws/portalfiles/portal/135470555/20181102_OvO_Eindverslag_Clephas.pdf

Kaczmarczyk, L. C., East, J. P., Herman, G. L., & Petrick, E. R. (2010). Identifying Student Misconceptions of Programming. *SIGCSE '10: Proceedings of the 41st ACM technical symposium on Computer science education* (pp. 107-111). New York, NY, United States: Association for Computing Machinery. doi:10.1145/1734263

Kallia, M., & Sentance, S. (2019). Learning to use Functions: The Relationship Between Misconceptions and Self-Efficacy. *SIGCSE '19: Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 752-758). New York, NY, United States: Association for Computing Machinery. doi:10.1145/3287324

Koopman, . (2017). *Diep leren : praktische handreikingen voor het bevorderen van diep leren bij leerlingen in het voortgezet onderwijs.* Eindhoven: Technische Universiteit Eindhoven. Opgehaald van https://research.tue.nl/files/76093257/NRO_praktijk_boek_Diep_Leren_opgemaakt_def.pdf

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

Novak, J. (2002). Meaningful learning: the essential factor for conceptual change in limited or inappropriate propositional., (pp. 548–571). doi:https://doi.org/10.1002/sce.10032

Plass-Oude Bos, D. (2015). *Identifying and Addressing Common Programming Misconceptions with Variables.* Twente: Universiteit Twente. Opgehaald van https://essay.utwente.nl/70455/1/Oude%20Bos%20Danny%20-%20S0021407%20-%20Afstudeerscriptie.pdf

Qian, Y., & Lehman, J. (2017, Oktober 27). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education*, pp. 1-24. doi:10.1145/3077618

Sekiya, T., & Yamaguchi, K. (2013, 11 14). Tracing quiz set to identify novices' programming misconceptions. *ACM International Conference Proceeding Series*, pp. 87–95. doi:10.1145/2526968.2526978

Smith III, J., diSessa, A., & Roschelle, J. (2009, 11 17). Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. *The journal of the learning science*(Volume 3), pp. 115–163. doi:10.1207/s15327809jls0302_1

Stulemeijer, W. (2010). *Misconcepties in het informatica-onderwijs.* Eindhoven: Eindhoven University of Technology. Opgehaald van https://research.tue.nl/files/46995141/693349-1.pdf

Swidan, A., Hermans, F., & Smit, M. (2018). Programming Misconceptions for School Students. *ICER '18: Proceedings of the 2018 ACM Conference on International Computing Education Research* (pp. 151-159). New York: Association for Computing Machinery. doi:10.1145/3230977.3230995

Taconis, R., & Terwel, J. (1999, 01). Strategisch omgaan met leerlingdenkbeelden in de exacte vakken: een praktisch overzicht en een theoretisch perspectief. *Tijdschrift voor didactiek der ß-wetenschappen, 1999*, pp. 91-109. Opgehaald van http://hdl.handle.net/1871/33187

TIOBE Software BV. (2021, 01 18). *TIOBE Index for January 2021*. Opgehaald van TIOBE Software BV: https://tiobe.com/tiobe-index/

Tyson, L., Venville, G., Harrison, A., & Treagust, D. (1997). A multidimensional framework for interpreting conceptual change events in the classroom. *Science Education*, pp. 387-404.

Verhagen, P., & Plomp, T. (1982). *Educational Technology and the new technologies.* Bucharest: European Centre for Higher Education-CEPES. Opgehaald van https://research.utwente.nl/files/266062837/Verhagen1989educational.pdf