Universiteit Leiden
The Netherlands

Opleiding Informatica

Hedy: De implementatie van een syntax highlighter

Omid Nasrat

supervisor:
Felienne Hermans

BACHELOR THESIS

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

12/07/2021

Abstract

Het leren van een programmeertaal wordt vaak gezien als een ingewikkeld proces. Om een programmeertaal toe te passen moet je het syntax herinneren en tegelijkertijd welgevormde zinnen met code schrijven. Vaak gaat er veel tijd verloren als er fouten in de geschreven code ontstaan, omdat het probleem niet gelijk zichtbaar is. Om programmeren (vooral voor kinderen) toegankelijker te maken, is Hedy ontwikkeld. Dit is een graduele programmeertaal bedoeld voor kinderen. In deze thesis is er gekozen om syntax highlighting in Hedy te ïmplementeren. Dit om het leerproces van kinderen te verbeteren. Syntax highlighting is het markeren van een stukje tekst of code, zodat het de leesbaarheid verbetert. In deze thesis wordt uitgelegd hoe een syntax highlighter in Hedy is geïmplementeerd. Ook wordt er uitgelegd hoe de syntax highlighter bepaalt wat er gemarkeerd moet worden per level in Hedy.

Contents

1 Introductie

De wereld ontwikkelt zich in een razendsnel tempo. In 1981 introduceerde IBM zijn eerste personal computer, ook wel bekend als PC. Door het grote succes werden de varianten laptops, tablets en mobiele telefoons vrij snel daarna uitgevonden. Dit zijn onmisbare apparaten geworden in de maatschappij die veel vergemakkelijking aanbieden in combinatie met het internet. Door deze apparaten kijk jij elke dag op Netflix naar je favoriete tv-serie, chat je via Whatsapp met je vrienden of lees je het nieuws. Door de digitalisering is het dus erg belangrijk om deze technologieën te begrijpen en ermee om te kunnen gaan. Voorheen was het niet nodig om eerst een software-update te geven aan je auto als er problemen waren, maar kon de monteur gelijk aan schroeven draaien.[6] De arbeidsmarkt is zodanig verandert waardoor een computer belangrijker is geworden. Dit heeft ertoe geleid dat de nieuwe generatie kinderen op een jonge leeftijd moeten leren programmeren, zodat ze voorbereid zijn op de toekomst.
Zo zijn er verschillende aanpakken bedacht om programmeren toegankelijker te maken. Een aantal aanpakken zijn in de vorm van een toy taal ontworpen, maar er zijn ook aanpakken waarbij een geheel nieuwe taal is ontworpen. Programma's in toy taal zijn ontwikkeld voor educatieve doeleinden. Het is geen complete taal, maar een taal specifiek gericht op een specifiek probleem. Hieronder zijn een aantal van die aanpakken uitgelegd.

- Apple[2] heeft Swift playgrounds ontwikkeld. Hiermee kunnen kinderen de taal Swift aanleren om te programmeren. Met Swift Playground kunnen er spelenderwijs problemen opgelost worden. Denk hierbij aan het maken van kleine puzzels en animatie. In figuur 1 is te zien hoe Apple kinderen duidelijk wil maken dat een poppetje kan lopen op basis van de methodes die aangeroepen woren links van het scherm.

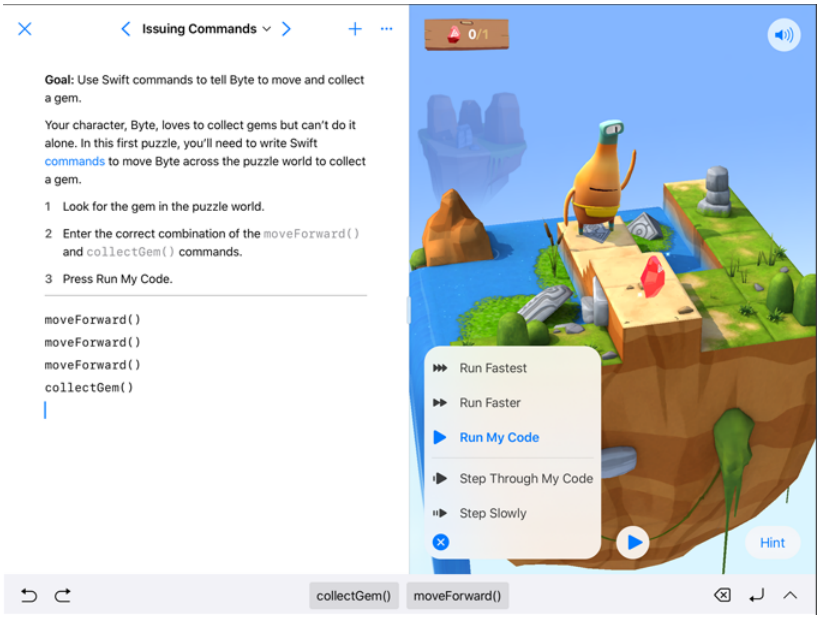

Figure 1: Swift Playgrounds, source: Swift

- Scratch[3] is een visuele programmeertaal verwerkt als een online webapplicatie, waarbij er aan de linkerkant van het scherm blokken van methodes zichtbaar zijn. Zodra je op een van

die methodes drukt, zie je het resultaat gelijk op de rechterkant van het scherm. In figuur 2 is de user interface zichtbaar met de mogelijke knoppen die gedrukt kunnen worden, waardoor er animatie plaatsvindt. Scratch focust zich op het maken van verhalen, games en animaties.

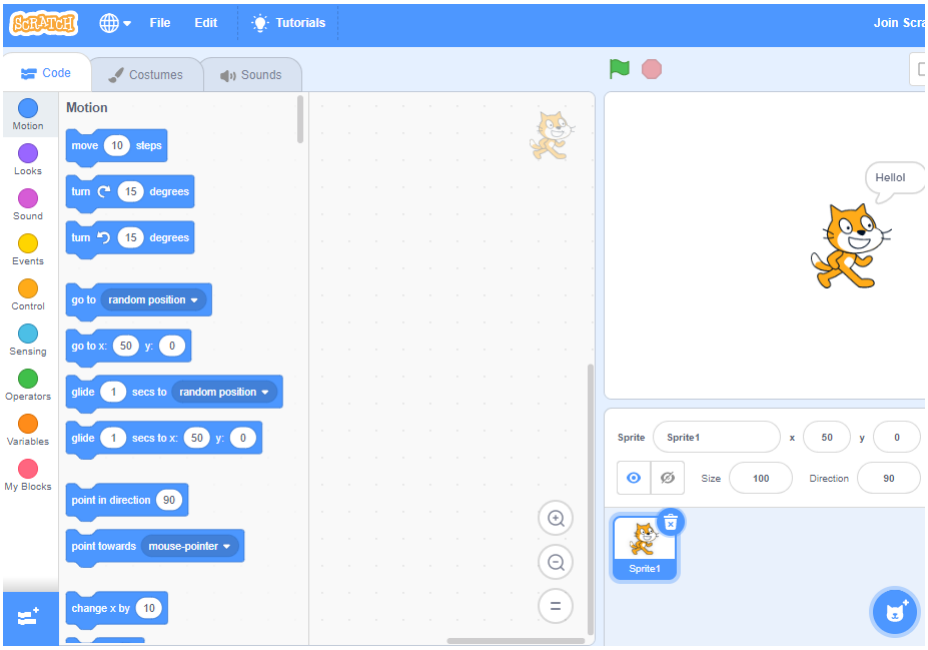

Figure 2: webapplication: Scratch, source: Scratch

In deze thesis stel ik een andere aanpak genaamd Hedy voor. Hedy is een graduele programmeertaal speciaal gericht op kinderen. Een graduele programmeertaal houdt in dat je stap voor stap een nieuwe taal leert. Door deze aanpak hoeven kinderen dus niet veel code te onthouden, waardoor de cognitieve druk op de hersenen kan verminderen. Om deze aanpak te vereenvoudigen heb ik ervoor gekozen om een syntax highlighter te implementeren. Dit zou ertoe moeten leiden dat het leerproces vergemakkelijkt wordt.

1.1 Contributies

Deze thesis zal de volgende items bevatten:

- Syntax highlighting voor elk level
- Een uitleg van de implementatie van een syntax highlighter voor elk level

2 Background

2.1 Hedy

Hedy is een open source graduele programmeertaal met het doel om programmeren in de Python taal makkelijker en toegankelijker te maken voor kinderen.[?] Hierbij is het niet nodig om al kennis

te hebben van programmeren. Tevens is het belangrijk om te vermelden dat Hedy gratis is.
Het leren van syntax in een programmeertaal is een lastige taak. Voor mensen is het lastig om veel informatie te onthouden zonder repetitie en/of het maken van relaties tussen de informatie. Voor het verbeteren van het leerproces wordt er gebruik gemaakt van een levelsysteem die gebaseerd is op het leveren van informatie in stappen. Als informatie in kleine delen verwerkt wordt, dan onthoud je het beter. Door gebruik te maken van levels, leren de gebruikers de syntax stap voor stap. Syntax is de manier van het plaatsen van woorden en zinnen, zodat een welgevormde zin ontstaat. In het allereerste level wordt aangeleerd hoe je moet printen, een vraag moet stellen en hoe je iets kunt herhalen wat al getypt is. In dit level leer je dus een versimpelde syntax van wat je uiteindelijk moet beheersen. Door deze manier van aanpak is het makkelijker voor kinderen om te leren programmeren, want ze worden niet overweldigd met veel nieuwe informatie.

2.2 Python

Python is een krachtige en een makkelijke programmeertaal, omdat het een versimpelde syntax gebruikt. Het vertoont veel similariteit met onze natuurlijke taal. Daarnaast bevat het een sterke community die constant aan Python werkt om het beter te maken, want Python is een open source taal. Daarnaast wordt het veel gebruikt bij de ontwikkeling van webapplicaties en voor data science. Dit zou niet mogelijk zijn zonder haar python libraries en frameworks die hierop gericht zijn. Een veelbekende library/framework die gebruikt wordt in data science is pandas, matplotlib en Numpy. Bekende libraries/frameworks voor web-development zijn Django en Flask. Door deze eigenschappen was Python de ideale programmeertaal om Hedy in te kunnen ontwikkelen.

2.3 Syntax highlighter

Syntax highlighting is een hulpmiddel gebruikt in text editors, waarbij delen van een stuk tekst verkleuren en/of de lettertype wijzigt volgens bepaalde regels om structuren te herkennen. Dit hulpmiddel maakt het schrijven van programmeercode makkelijker, want er kan visueel opgemerkt worden of er fouten in de programmeercode zijn. Structuren verkleuren en fouten verkleuren niet terwijl je wel verwacht dat ze verkleuren. Door deze verbetering kan er ook minder tijd verloren gaan met het zoeken naar het probleem.
Door het implementeren van een syntax highlighter verwacht ik een verbetering in het leerprocess tijdens het gebruik van Hedy. Want uit onderzoek is gebleken dat wiskundige problemen makkelijker op te lossen zijn met visuele strategieën[5]. Door het kleuren van stukjes tekst verwachten wij een verbetering in de leerprocess, want het kleuren van stukjes tekst maakt het visueel makkelijker. Om de implementatie van een Syntax highlighter mogelijk te maken, heb ik gebruik gemaakt van de geïmplementeerde ACE editor in Hedy.

2.4 ACE editor

Ace is een open source embeddable code editor geschreven in de taal Javascript. Embeddable houdt in dat het ingebouwd is in een ander component, in dit geval een webbrowser. Ace editor beschikt over dezelfde functies als Sublime of Vim. Dit zijn text editors die niet geïmplementeerd zijn in web-applicaties. Hierbij kan je denken aan de standaard kladblok applicatie in Windows, maar dan

met extra functionaliteiten waar veel programmeurs gebruik van maken. Bekende functionaliteiten die ACE aanbiedt zijn:

- Syntax highlighting voor meer dan 110 talen;
- 20 thema's;
- Automatische indentatie;
- Code folding: het verbergen en zichtbaar maken van code in de editor
- Highlighten van matchende haakjes in verschillende vormen;
- Knippen, kopieren en plakken.

2.5 State machine

Om gebruik te maken van syntax highlighting in de ACE editor moet het concept van state machines begrepen worden. Een state machine is een model waar gebruik wordt gemaakt van een eindig aantal toestanden. Vanuit elke toestand zijn er een bepaald aantal transities mogelijk door een invoer. Deze transities kunnen leiden naar een accepterende toestand of naar een andere toestand. Als je in een accepterende toestand komt, dan accepteer je de invoer. Een simpel voorbeeld is het indrukken van een licht knop, dat zichtbaar is in figuur 3. Als het licht uitstaat en je drukt op de

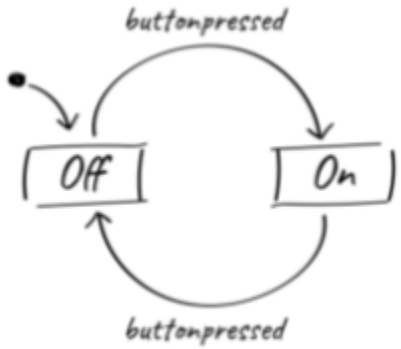

Figure 3: light button state machine. Source: itemis.com

knop, dan ga je naar de toestand waar het licht uitstaat. Het tegenovergestelde geldt als het licht aanstaat, dan gaat het licht uit.
Voor de implementatie van de Syntax highlighter heb ik gebruik gemaakt van bovenstaand concept in javascript code. In deze implementatie heb ik ook gebruik gemaakt van reguliere expressies.

2.6 regex

Reguliere expressies, ookwel bekend als regex, is een manier om patronen te beschrijven.[7] Hierdoor is het mogelijk dat een computer softwarematig bepaalt wat er met het gevonden stukje tekst moet gebeuren. Door dit eigenschap is regex ideaal om te gebruiken voor een syntax highlighter. Het

markeren van een stukje tekst in een groter stuk tekst moet plaats vinden door patronen.
Een reguliere expressie is een stukje tekst dat kan bestaan uit cijfers, letters en tekens. De tekens in een reguliere expressie beschrijven een specifieke patroon. Hieronder staan een aantal veelgebruikte tekens beschreven:

- ^ : Dit teken geeft aan dat een bepaald patroon moet starten aan het begin van een zin. Een voorbeeld is: "^a". Dit betekent dat een zin moet starten met de letter a.
- $: Dit teken geeft aan dat het patroon moet eindigen aan het eind van een zin. Een voorbeeld is: "a$". Dit houdt in dat de zin moet eindigen met een letter a aan het eind.
- + : Dit houdt in dat alles wat hiervoor stond tenminste een keer moet voorkomen. Een voorbeeld is: "t+". Hierdoor wordt invoer als tekst en tak geaccepteerd.
- ? : Dit houdt in dat alles wat hiervoor staat maximaal 1 keer mag voorkomen. Een voorbeeld is: "t?". Hierdoor wordt invoer als tak, aap en top geaccepteerd, want er is maar maximaal 1 voorkomen van de letter t in het woord.
- * : Dit houdt in dat alles wat hiervoor staat nul of meerdere keren mag voorkomen. Een voorbeeld is: "a*". Deze reguliere expressie accepteert een lege reeks of een reeks met 1 of meer a's opeenvolgend.

2.7 Related Work

In het verleden zijn veel onderzoeken uitgevoerd om te controleren of het kleuren van stukjes tekst invloed had op de leesbaarheid. Deze onderzoeken hebben op verschillende manier plaatsgevonden. Zo is er een verband geobserveerd tussen syntax fouten met programmeer- en muziekervaring in de Sonic Pi source code met 10 testpersonen van ongeveer 12 jarigen[1]. Hieruit bleek dat de tijd tot het afmaken van een taak verminderde bij onervaren programmeurs. Daarnaast is er bij een andere onderzoek van 10 testpersonen gebleken dat de tijd tot het afmaken van een taak verminderde bij onervaren programmeurs en het aantal context switches ook. Context switches is het aantal keer dat een participant naar een ander stukje tekst kijkt dat anders is dan het stukje ervoor. Dus context switches is het aantal keer kijken naar een ander stukje tekst[4].

3 Vereisten

Ik heb gebruik gemaakt van Pycharm om bij te kunnen dragen aan Hedy. Pycharm is een Integrated development environment, ookwel bekend als IDE. Dit is software bedoeld om applicaties op te bouwen in de programmeertaal van Python. Daarnaast heb ik gebruik gemaakt van Github. Github is een versie controle platform wat vooral gebruikt wordt door ontwikkelaars. Github geeft ontwikkelaars de mogelijkheid om met meer mankracht aan hetzelfde project te werken zonder dat er problemen onstaan. Tenslotte heb ik een javascript file aangemaakt om de syntax highlighter in te implementeren.

3.1 Hedy klonen

Hedy is een open source graduele programmeertaal wat op Github staat. Hierdoor is iedereen in staat om vrijwillig bij te dragen aan Hedy. Hiervoor is de eerste stap het klonen van dit project via Github. Daarna kan het geopend worden met een IDE als Pycharm of met een texteditor.

3.2 Code uitvoeren

Nadat de wijzigingen zijn doorgevoerd aan de code kan dit gelijk getest worden. Allereerst zoek je het bestand App.py op. Daarna kan er via Pycharm simpelweg op het play knopje gedrukt worden. Deze staat rechtsboven in de IDE. Dit is een groen driehoekje met een draai van 90 graden, ook te zien in figuur 4. Zodra het programma in uitvoering is, zie jij als gebruiker niets gebeuren. Dit

Figure 4: uitvoer knop in Pycharm

komt omdat het een achtergrond proces is. Dit proces laat Hedy uitvoeren in de local host op je webbrowser. Local host houdt in dat jouw computer de lokale computer is die het programma host. Hierdoor is het programma dat gehost wordt zichtbaar op het host adres. Het host adres om Hedy zichtbaar te maken op de webbrowser is: *localhost* : 5000, hiervoor is het niet nodig om www te typen voor het adres. In figuur 5 wordt duidelijk gemaakt waar in een webbrowser het adres getypt moet worden.

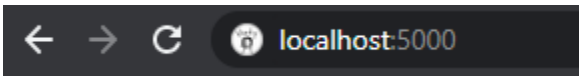

Figure 5: url balkje op je webbrowser naar keuze

3.3 testen

Om de code te testen, moet je het adres van figuur 7 in je webbrowser bezoeken. Hierna kan je het stukje van de website bezoeken waar jij aanpassingen aan hebt gebracht. Hierdoor kan je controleren of alles volgens verwachting werkt. In het geval van syntax highlighting moet je gelijk op het "probeer het uit" knopje drukken. Hierdoor start je gelijk bij het eerste level van Hedy.

3.4 Hedy user interface

Hedy haar user interface is als volgt ingedeeld, zoals te zien op figuur 6:

- Aan de linker kant staan de nieuwe commando's die geleerd moeten worden. Als je erop drukt krijg je een voorbeeld invoer. Hierop is zichtbaar hoe het gebruikt moet worden.
- Bovenaan staan er tabs met kleine opdrachten, waarin de nieuw geïntroduceerde commando's toegepast worden.

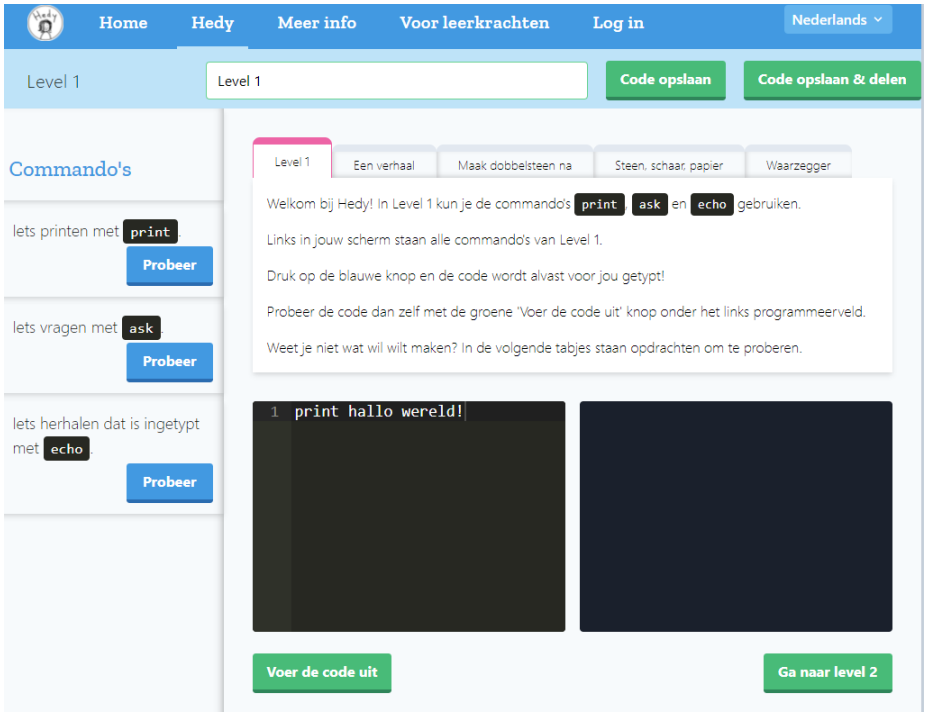

Figure 6: visualisatie van het eerste level van Hedy zonder syntax highlighting

- twee donker gekleurde rechthoeken. De linker rechthoek wordt gebruikt als een editor. Hierin kan je code typen. Op de rechter rechthoek zie je het gevolg van jouw code.
- Onderaan staan er twee of drie groene knopjes. Er is een knop voor het uitvoeren van de code. En situationeel zijn er twee knoppen, waarmee je van levels kan veranderen. Of is er een knop om naar de vorige level te gaan.

4 Implementatie van de syntax highlighter

Tijdens de implementatie van de syntax highlighter is er voor elke level een stukje code toegevoegd in de syntaxModeRules.js bestand. Hierin staan de regels beschreven voor de highlighter. Door deze regels kan de highlighter het juiste stukje code markeren in een bepaalde kleur. De regels zijn als volgt op te schrijven:

- Kies de toestand waar de regels toegepast mogen worden. In het geval dat je een nieuwe toestand nodig hebt, maak die aan.
- Declareer binnen de toestand een token, regex en eventueel een next.
- met de token soort geef je aan in wat voor kleur het gemarkeerd moet worden.
- regex: op basis van welke patroon
- next: transitioneer naar de gegeven toestand

- defaulttoken: standaard token die gebruikt moet worden als het geen van de tokens is.

Voordat deze regels toegepast kunnen worden door Hedy moet er eerst een aanpassing plaatsvinden in het bestand app.js. In dit bestand is er een boolean variabele aangemaakt om Hedy te vertellen of de syntax highlighter zijn werk mag verrichten of niet. Een boolean variabele houdt in dat de variabele waar(True) of niet waar(False) is. Daarnaast is er voor elk level specifiek vermeld welke highlighter gebruikt moet worden. Er is dus een syntax highlighter voor elk level apart, omdat elk level op zich een eigen taal is. Hedy is namelijk een graduele taal.

4.1 level 1

In het eerste level worden er drie commando's geintroduceerd:

- Print
- Ask
- echo

Als gebruiker kan je het commando print als volgt gebruiken: Allereerst typ je het commando print en vervolgens typ je iets erachteraan. Het resultaat is dan dat je geprint krijgt dat wat achter het substring "print" staat. Om de commando print te highlighten heb ik dus gebruik gemaakt van de volgende regel: als print aan het begin van een zin staat, dan moet dit gehighlight worden, want alles erna is een stukje tekst. Het resultaat van deze regel is dan figuur 7 en 8.
De commando "ask" wordt gebruikt om een vraag te stellen aan de gebruiker. En de commando

```
1 hallo print Hallo welkom bij Hedy!
```

Figure 7: print commando niet aan het begin

```
1 print Hallo welkom bij Hedy!
```

Figure 8: print commando aan het begin

"echo" wordt gebruikt om te herhalen wat een gebruiker eerder heeft beantwoord op een vraag die gesteld was door de commando "ask". Voor deze twee commando's geldt dezelfde regel als voor die van print. Als het aan het begin van de zin staat, dan worden ze gemarkeerd door de syntax highlighter. De state machine voor dit level bevat dan ook maar 1 toestand, de starttoestand. De regels code van deze level zijn zichtbaar in figuur 9. Elke transitie vanuit deze toestand zorgt ervoor dat je weer naar de start toestand komt. Echter zorgt elke transitie ervoor of een bepaald stukje regex code gemarkeerd wordt of niet.

4.2 level 2

In level 2 zijn er twee nieuwe commando's geïntroduceerd en zijn er extra functionaliteiten toegevoegd aan bestaande commando's:

- "is" met dit commando kan je aan een woord een waarde toekennen. Oftewel, je kan variabelen maken. Een voorbeeld kan zijn: Naam is Hedy
  Naam heeft nu de waarde Hedy als dit geprint wordt met de commando "print". Print kan vanaf nu dus ook de waarden van variabelen printen.

```
"start": [{
  token: "keyword",
  regex: "^print "
},{
    token: "keyword",
    regex: "^ask "
},
{
    token: "keyword",
    regex: "^echo "
},{
    token: "comment",
    regex: "#"
}],
```

Figure 9: Javascript en regex code van de state machine van level 1

- de commando "is" kan in combinatie gebruikt worden met "ask". Hierdoor kan de gebruiker een waarde toekennen aan een variabele tijdens het uitvoeren van de code.
- met "is" kan je een lijstje maken. Een voorbeeld kan zijn: dieren is hond, aap, kat. De print commando kan nu dus ook lijstjes printen in de vorm van: ['a','b', ... 'z']
- is er een commando "at random" beschikbaar. Dit commando wordt in combinatie met een lijst gebruikt. Hierdoor kan je een random waarde uit een lijst printen met de print commando.

In dit level moeten deze commando's dus correct gehighlight worden. Hiervoor zijn er in de code voor dit level drie state machine toestanden gemaakt. Start, print rest en rest. De state machine begint in de toestand start. Vanuit deze toestand kan er getransitioneerd worden naar een ander toestand of naar hetzelfde toestand. Het is belangrijk om te vermelden dat oude syntax highlighter code soms mee wordt genomen naar nieuwe levels. Zo is in dit level de manier van de commando "print" highlighten niet veranderd en wordt dit nogsteeds gecontroleerd vanuit de start toestand. Echter kan het transitioneren naar de toestand "print rest" als de commando "at random" tussen de zinnen ziet staan of aan het eind. Als dit het geval is, begint de highlighter weer in de start toestand met zoeken naar woorden om te markeren. Tenslotte controleert de highlighter vanuit de start toestand of het de woord(en): "is" of "is ask" ziet. In dit geval wordt dit gemarkeerd en getransitioneerd naar de rest toestand. Deze toestand is gemaakt zodat er geen herhaling plaatsvindt van het markeren van "is" of "is ask" in een enkele zin. Dit is te zien in figuur 10.

```
1 kleur is ask Wat is je lievelingskleur?
```

Figure 10: "is" na "is ask" wordt niet gemarkeerd in de rest toestand

4.3 level 3

Vanaf dit level is de "print" commando verandert. Om een stukje tekst te printen, moet er vanaf nu gebruik worden gemaakt van enkele aanhalingstekens. Maar om de waarde van een variabele te printen die je gemaakt hebt via de commando "is" hoef je geen aanhalingstekens te gebruiken. Dit is te zien in figuur 11. De implementatie van dit level is bijna hetzelfde als die van level 2. Voor dit

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

Figure 11: printen van een variabele en printen van een stukje tekst met aanhalingstekens

level is er een nieuwe toestand gemaakt in de state machine genoemd "print option". Zodra de start toestand de substring "print" ziet, dan transitioneert het naar de "print option" toestand. Het doel van deze toestand is het herkennen van een print commando waar er een stukje tekst geprint moet worden met aanhalingstekens. Als er een enkele aanhalingsteken gevonden wordt, dan transitioneert het naar de "print rest" toestand. Deze toestand is bijna hetzelfde gebleven, echter wordt hier weer gecontroleerd op een enkele aanhalingsteken. Als dit het geval is, dan weet de syntax highlighter dat het dit stukje tekst moet markeren in een andere kleur om aan te geven dat dit het tekst is wat geprint moet worden.

4.4 level 4

In level 4 zijn twee nieuwe commando's geïntroduceerd:

- If. De if statement is een commando wat gebruikt wordt om een keuze te maken tussen verschillende opties
- Else. De else statement wordt in combinatie met een if statement gebruikt. Het is om aan te geven dat als geen van de opties genomen wordt, dat dan de instructie na de else statement uitgevoerd moet worden.

In dit level wordt de bestaande code van level 3 hergebruikt met een kleine aanpassing. Voor de if en else statement is er een aparte toestand in de state machine. Deze toestand is te bereiken via de start toestand waar gecontroleerd wordt of een if statement aan het begin van een zin staat en of er een else statement tegengekomen wordt na een if statement.
Als de highlighter dit herkent, dan transitioneert het naar de "ifElseSpace" toestand. Vanuit deze toestand wordt er rekening gehouden dat alles binnen een if of een else statement plaatsindt. Nadat het klaar is transitioneert het terug naar de start toestand.

4.5 level 5

De methode "repeat" wordt in dit level geïntroduceerd. Repeat heeft als doel dat je code meerdere keren kan herhalen. Deze methode is een opstapje richting het gebruik van de for loop. Een for loop is een geavanceerde commando wat hetzelfde doet, maar dan met meer opties. Hierdoor bevat het meer controle, maar is het iets complexer voor een kind. In figuur 12 is geïllustreerd wat de uitvoer

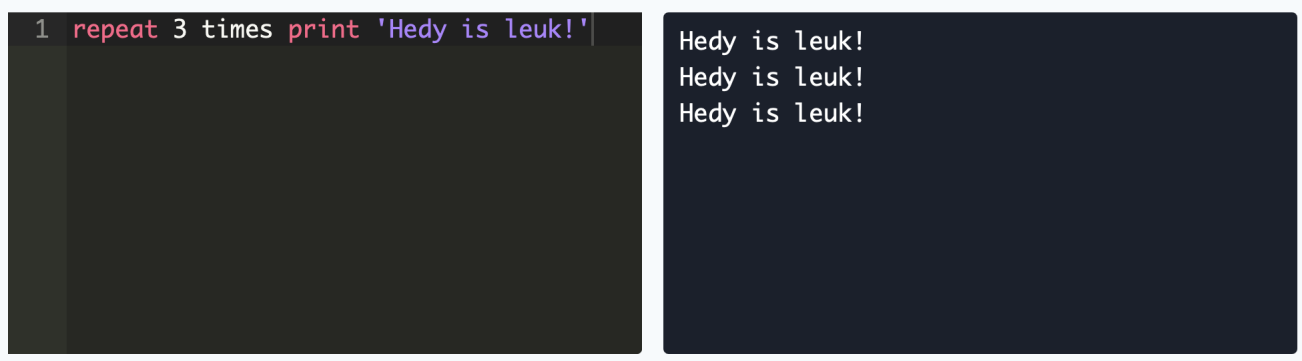

Figure 12: output van de repeat commando

is van het gebruik van de repeat commando. Voor de implementatie van dit level is er gebouwd op het vorige level. In de starttoestand is het herkennen van de repeat commando aan het begin van de zin geïmplementeerd. Tenslotte is het herkennen van de repeat commando ook verwerkt in de ifElseSpace toestand. Hierdoor is het mogelijk repeat te herkennen binnen if of else statements.

4.6 level 6

Vanaf dit level is rekenen geïntroduceerd. Zo is het mogelijk nu gebruik te maken van het plus en min teken. Daarnaast is het ook mogelijk om te vermenigvuldigen. Echter is er niet een "keer" symbool op het toetsenbord. Hierdoor wordt er gebruik gemaakt van de "*" symbool, wat te zien is op figuur 13. Rekenen is in twee toestanden verwerkt. In de start toestand, want het kan gebruikt

Figure 13: keer symbool vervangen door een *

worden bij het toekennen van een waarde aan variabele, zoals $a = 5 * 5$. Daarnaast kan het ook gebruikt worden binnen een if of een else statement. Een voorbeeld zou kunnen zijn: if a is 5 print 'a = 5'.

4.7 level 7

Level 7 introduceert de mogelijkheid om meerdere regels samen te herhalen. Dit is mogelijk gemaakt door in te springen in regels met 4 spaties. Door middel van het inspringen leren we kinderen aan dat de ingesprongen regels onderdeel zijn van de zin erboven. In figuur 14 wordt duidelijk gemaakt dat regel 4 aangeroepen wordt door regel 3 en regel 3 zelf wordt aangeroepen als de if conditie waar is in regel 2. Deze toevoeging vereist echter geen aanpassingen bij de syntax highlighter. Alle patronen worden herkend en gemarkeerd volgens de regels die tot nu toe geïntroduceerd zijn.

4.8 level 8

Vanaf dit level wordt repeat vervangen door een nieuwe commando: for i in range x to x, ook bekend als een for loop. Hierbij is x $\in \mathbb{N}$, in andere woorden x is een gehele cijfer in de natuurlijke

```
1 kleur is ask Wat is je lievelingskleur?
2 if kleur is groen
3     repeat 3 times
4         print 'mooi!'
5 else
6     repeat 5 times
7         print 'niet zo mooi'
```

Figure 14: inspringen in regels zichtbaar gemaakt

getallen. Door deze nieuwe notatie schrijf je dus nu: for i in range 1 to 5 in plaats van repeat 5 times. Na het gebruik van deze nieuwe commando moet er verplicht ingesprongen worden. Door

```
1 for i in range 1 to 10
2     print i
3 print 'Wie niet weg is is gezien'
```

Figure 15: for loop

het inspringen in figuur 15 in regel 2 geef je aan dat alleen die print in regel 2, 10 keer uitgevoerd moet worden. Dus de uitvoer zal zijn dat het programma tot 10 zal tellen en vervolgens 'Wie niet weg is is gezien' print. Aangezien dit een nieuwe commando is die een oude commando vervangt, hebben we eerst alle patronen waarbij repeat gebruikt werd gewist. Vervolgens zijn er drie nieuwe toestanden gemaakt.

- forloop1
- forloop2
- forloop3

Vanuit de starttoestand wordt gecontroleerd of er een "for" gelezen wordt aan het begin van een zin. Als dit het geval is transitioneert de syntax highlighter naar de toestand "forloop1". In deze toestand wordt gecontroleerd of "in range" gelezen wordt. Als het herkent wordt, dan transitioneert het naar de toestand "forloop2". Vervolgens is er één laatste controle die op zoek gaat naar "to". Vergelijk als de vorige toestand transitioneert dit naar een nieuwe toestand genaamd "forloop3". Vanaf deze toestand leest het alle invoer als normale invoer en gaat het aan het einde van de regel naar de start toestand. In figuur 16 zijn de 3 nieuwe toestanden en hun transities zichtbaar.

4.9 level 9

In dit level vinden er een aantal kleine aanpassingen plaats en wordt er een nieuwe commando toegevoegd. De for loop die in level 8 beschreven is, wordt aangepast met een ":" aan het eind. Hierdoor moet het gelijk duidelijk zijn dat er ingesprongen moet worden, omdat je het einde van de for commando bent tegengekomen. Dezelfde aanpassing vindt plaats in de if en else statement. Beide commando's worden afgesloten met een ":" aan het einde van de regel. Tenslotte is de

```
"forloop1": [{
  token: "keyword",
  regex: " in range ",
  next: "forloop2"
},{
  defaultToken : "text"
}],

"forloop2": [{
  token: "keyword",
  regex: " to ",
  next: "forloop3"
},{
  defaultToken : "text"
}],

"forloop3": [{
  token: "text",
  regex: "$",
  next: "start"
},{
  defaultToken : "text"
}],
```

Figure 16: for loop toestanden en haar transities in de code

commando "elif" toegevoegd. Elif is een commando die vergelijkbaar is met "else if" van de programmeertaal C++. Deze commando wordt gebruikt als de huidige if commando niet waar is, dan wordt gecontroleerd of de volgende if waar is etc. Als ze allemaal niet waar zijn kom je bij de else commando terecht. Een voorbeeld is zichtbaar in figuur 17. Voor de implementatie van ":" in

```
1 a is 2
2 if a is 1:
3     print 'a is 1'
4 elif a is 2:
5     print 'a is 2'
6 else:
7     print 'a is niet 1 of 2'
```

Figure 17: elif commando geïllustreerd

de commando's hoeven er geen aanpassingen plaats te vinden in de syntax highlighter. Er is voor gekozen om de ":" niet te markeren net als in de Python taal. De enige aanpassing in de syntax highlighter is de toevoeging van de herkenning van de elif commando in de start toestand.

4.10 level 10

In dit level is het mogelijk om de for commando te plaatsen in een andere for commando. Dit wordt ook wel for loop nesting genoemd. Dit is vooral handig als je meerdere iteraties nodig hebt over een stukje code. Het wordt vooral gebruikt in situaties als het vergelijken van een lijst met woorden met een andere lijst met woorden om te controleren of ze hetzelfde zijn. Ook de if statement heeft de functionaliteit van nesting gekregen. In de syntax highlighter hebben er geen wijzigingen plaatsgevonden, omdat er al in elke regel gecontroleerd wordt of er een for loop of is statement herkend wordt. Dit is te zien aan de start toestand in de code.

4.11 level 11

Vanaf dit level wordt de ask commando vervangen door de python vergelijkbare commando input, hierbij maakt deze commando gebruik van haakjes. Daarnaast wordt er vanaf nu ook gebruik

gemaakt van haakjes bij de print en for commando. Voorbeeld invoer is te zien in figuur 18. Voor

Figure 18: haakjes in de for, input en print commando met uitvoer rechts

de implementatie van de syntax highlighter veranderen dan een aantal dingen.

- Zo wordt in de toestand forloop1 gecontroleerd op een open haakje aan het einde. In toestand forloop2 wordt gecontroleerd of er een gesloten haakje tegenkomen wordt. Als dit het geval is, dan transitioneert de state machine weer naar de start toestand. Toestand forloop3 is vanaf heden overbodig en is dus weggehaald.

- De nieuwe print commando zorgt ervoor dat we vanaf nu controleren op een open haakje in de start toestand. En in de print option toestand wordt er gecontroleerd op een gesloten haakje om terug te transitioneren naar de start toestand.

- Tenslotte is de ask commando vervangen door input. Hierbij wordt er in de start toestand gecontroleerd op de invoer "input" met een open haakje. Hierdoor vindt er een transitie plaats naar een nieuwe toestand genaamd "input". Vanuit deze toestand kan er getransitioneerd worden naar een toestand "input rest" als er een enkele aanhalingsteken herkend wordt. Vanuit die toestand kan er een transitie plaatsvinden naar "input eind". Dit vindt plaats als er weer een enkele aanhalingsteken plaatsvindt. Tenslotte als er een gesloten haakje gelezen wordt, dan transitioneert het terug naar de start toestand om ook andere patronen te kunnen herkennen.

4.12 level 12

In dit level worden een aantal dingen toegevoegd aan de lijst functionaliteit:

- Voortaan moet er gebruik worden gemaakt van haakjes om duidelijk te maken dat we te maken hebben met een lijst. Daarnaast is het ook verplicht om voortaan enkele aanhalingstekens te gebruiken om items in een lijst te plaatsen. Hierdoor is er meer similariteit met de Python taal.

- Vanaf nu is het ook mogelijk om gelijk een item aan te wijzen in een lijst. Dit wordt mogelijk gemaakt door vierkante haakjes te gebruiken met een nummer ertussenin.

- Daarnaast is het mogelijk om ook random een item uit de lijst te pakken. Dit wordt gedaan als volgt: fruit[random], hierdoor wordt er random een fruit gepakt uit de lijst.

```
1 fruit is ['banaan', 'appel', 'kers']
2 randomfruit is fruit[random]
3 print(randomfruit)
```

Figure 19: de drie nieuwe lijst functionaliteiten

Het gebruik van deze aanpassingen zijn terug te zien in figuur 19. Alleen de functionaliteit waarbij je een lijst aan moet geven met haakjes, kan worden verwerkt in een syntax highlighter. De overige twee aanpassingen vereisen geen implementatie in de highlighter. De aanpassing die plaats heeft gevonden in de syntax highligter is in het geval dat de commando "is" gedetecteerd word. In de vorige levels werd er niet getransitioneerd, maar vanaf dit level wel. Het is immers niet zeker of er na detectie een lijst komt of niet. Daarom wordt er getransitioneerd naar de toestand: "LijstOfGeenLijst". In deze toestand wordt er gecontroleerd of er een open haakje gezien wordt. Als dit het geval is, dan transitioneert het naar de toestand: "Lijst". Daarna wordt er gecontroleerd of er aanhalingstekens zijn, dus items die in de lijst zitten. In dat geval transitioneert het naar een toestand: "singleQuote". Deze toestand geeft aan dat je weer een aanhalingsteken kan verwachten, want er wordt een item gelezen. Na het lezen transitioneert het terug naar de toestand "Lijst". Tenslotte als er een gesloten haakje gezien wordt, dan weet de state machine dat het klaar is met het lijstje die ingelezen werd en transitioneert het naar de start toestand om nieuwe commando's te herkennen.

4.13 level 13

In dit level is er een nieuwe functionaliteit toegevoegd: Booleans. Hierdoor kan je variabelen dus op waar of niet waar zetten. Dit kan handig zijn als je bij wil houden of je iets gedaan hebt of niet, waardoor je de juiste keus kan maken. Een voorbeeld is te zien in figuur 20. Door middel

```
1  goed_antwoord is False
2  antwoord is input('Wat is 5*5')
3  if antwoord is 25:
4      goed_antwoord is True
5  else:
6      goed_antwoord is False
7  if goed_antwoord is True:
8      print('Dat is goed!')
9  if goed_antwoord is False:
10     print('Dat is fout!')
```

Figure 20: voorbeeldcode met het gebruik van True en False

van deze functionaliteit kan je een variabele op waar(True) of niet waar(False) zetten. Dit kan dus plaatsvinden na de commando "is". Ook kan je booleans gebruiken in if en else statements om te kijken of een conditie waar is of niet waar is. Daarom is dit in de syntax highlighter als

volgt geïmplementeerd: In het geval van de commando "is" wordt er eerst getransitioneerd naar de "LijstOfGeenLijst" toestand. Daarna wordt er gecontroleerd of er True of False gedetecteerd wordt. Als dat het geval is, wordt het gemarkeerd. In het geval van een if of else statement moet er dus vanaf de "ifElseSpace" toestand gecontroleerd worden of er weer een commando "is" komt. Vergelijkbaar als hiervoor controleer je of er een True of False komt.

4.14 level 14

Meerdere if statements onder elkaar is na dit level niet meer nodig. Vanaf deze leeftijd worden de "and" en "or" operanden geintroduceerd. Deze operanden kunnen binnen een enkele if statement gebruikt worden om meerdere condities met elkaar te vergelijken. Beide operanden hebben hun eigen waarheidstabel te zien in tabel 1. In figuur 21 en 22 is te zien dat de and operand ervoor

	0	1
0	False	False
1	False	True

AND

	0	1
0	False	True
1	True	True

OR

Table 1: Waarheidstabel And en Or operanden

```
1 antwoord1 is 5
2 antwoord2 is 4
3 if antwoord1 is 5 and antwoord2 is 4:
4     print('antwoorden zijn goed!')
```

Figure 21: and operand in een if statement

```
1 antwoord1 is 5
2 antwoord2 is 4
3 if antwoord1 is 5:
4     if antwoord2 is 4:
5         print('antwoorden zijn goed!')
```

Figure 22: geen gebruik van de and operand

zorgt dat de functionaliteit hetzelfde blijft, maar dat er minder regels code nodig zijn ten op zichte van het gebruik zonder. De implementatie van deze operanden heeft plaats gevonden binnen de "ifElseSpace" en de "LijstOfGeenLijst" toestanden. Een and of or operand kan voorkomen binnen een if statement en binnen een if statement kan de commando "is" herkend worden. Zodra dat het geval is kan er weer gecontroleerd worden op een and of or operand. In zowel de toestand "ifElseSpace" en "LijstOfGeenLijst" vinden er geen transities plaats naar een andere toestand in het geval dat een and of or operand herkend wordt.

4.15 level 15

Hoe groter en ingewikkelder een programma wordt, des te belangrijker het plaatsen van commentaar wordt. Het plaatsen van commentaar zorgt ervoor dat later niet de hele code opnieuw gelezen moet worden. Je weet immers wat er gebeurt in een specifieke gedeelte van de code. Vanwege de handigheid is er in dit level de commentaar symbool geïntroduceerd. De gebruiker kan door het plaatsen van de symbool # aangeven dat alles in deze regel commentaar is en dat het genegeerd kan worden. Commentaar wordt dus nooit uitgevoerd en dat is de reden dat er in de syntax highlighter voor gekozen is om dit grijs te maken. In figuur 23 is te zien dat comments niet mee wordt genomen in de uitvoer. Om comments te implementeren in de syntax highlighter wordt er

Figure 23: code links en uitvoer rechts

simpelweg gecontroleerd in de start toestand of er een # herkend wordt. Als dat het geval is, dan wordt alles erna gezien als commentaar.

4.16 level 16

In het geval van condities is er veel meer dan alleen maar controleren of iets gelijk is aan elkaar. Zo moet er ook de mogelijkheid zijn om te controleren of een variabele of waarde kleiner of groter is dan de ander. Dit is de reden dat de kleiner dan en de groter dan operand is geïntroduceerd , < en > respectievelijk. Om de syntax highlighter dit te laten herkennen is er in de "ifElseSpace" toestand een controle op deze twee symbolen. In beide gevallen vindt er geen transitie plaats naar een ander toestand.

4.17 level 17

In dit level introduceren we de while loop. De while loop krijgt een statement mee die waar of niet waar is. Zolang het statement niet verandert, blijf je code uitvoeren die binnen die loop zit. Het verschil tussen deze loop en de for loop is dat je in de for loop een vast aantal keer itereert. Een while loop daarentegen kan nooit stoppen, als de statement niet van waarde verandert. In figuur 24 is een voorbeeld te zien van een while loop die wel stopt en in figuur 25 is er een voorbeeld van een while loop die nooit stopt. In figuur 24 verandert de tel variabele en uiteindelijk is de waarde groter

Figure 24: while loop

Figure 25: oneindige while loop

of gelijk aan 3, waardoor het stopt. In figuur 25 verandert de tel waarde nooit, hierdoor komt het nooit uit de while loop in regel 3. Om de while in de syntax highlighter te implementeren is er een nieuwe check aangemaakt in de start toestand. Als het een while loop herkent dan transitioneert het naar de "ifElseSpace" toestand. de mogelijkheden vanuit de while loop zijn hetzelfde als een if of een else statement in de syntax highlighter. Dit is de reden dat het naar de "ifElseSpace" toestand transitioneert.

4.18 level 18

Vanaf dit level is het ook mogelijk om een specifieke item uit de lijst te pakken. Om dit te doen, typ je de lijst naam en open je een haakje. Vervolgens typ je het item nummer in de lijst en sluit je het af met een haakje. In figuur 26 staat er een stukje code die het gebruik van deze functionaliteit illustreert. Voor de syntax highlighter vonden er in dit level geen aanpassingen plaats. Hierdoor

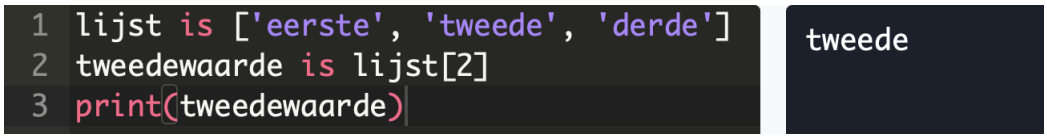

Figure 26: het printen van een specifieke lijst item

beschouwt de syntax highlighter de vorige level en dit level als één.

4.19 level 19

In dit level is een kleine functionaliteit toegevoegd voor lijsten. Door de methode length() kan je de lengte van een lijst bepalen. Als er drie items in de lijst staan, dan returnt de methode length drie terug. In figuur 27 wordt het gebruik van methode length() geïllustreerd. Om deze methode

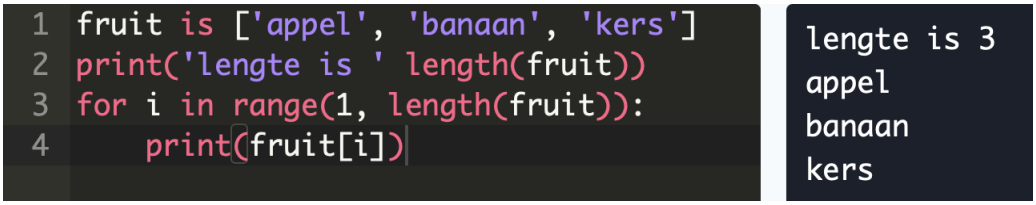

Figure 27: gebruik van de length methode met de uitvoer rechts

te herkennen in de syntax highlighter, zijn er veel aanpassingen aangebracht. Zo moet de length methode herkent worden in een for loop. Length kan hier gebruikt worden als variabele. Hierdoor kan je bepalen hoeveel iteraties een for loop moet itereren. Daarnaast moet deze methode ook herkent worden in een print commando. Tenslotte kan deze methode ook herkend worden in een if statement.

4.20 level 20

de commando "is" wordt vanaf dit level verwijdert en vervangen door 2 nieuwe operanden:

- =
- ==

Een enkele '=' teken geeft aan dat de waarde links verandert naar de waarde rechts van het teken. Als het teken twee keer achter elkaar geplaats wordt, dan betekent het dat de waarde links van die tekens gelijk moet zijn aan de waarde rechts van de tekens. Dus met het "==" symbool controleer je gelijkheid. In figuur 28 is een voorbeeld met een enkele en dubbele '=' symbool. In regel 2 verandert

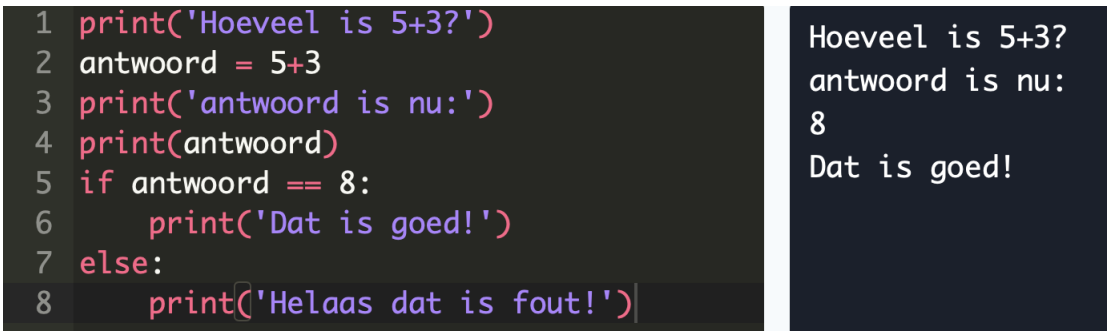

Figure 28: gebruik van '=' en "==" met de uitvoer rechts

de waarde van antwoord naar acht en in regel 6 wordt er gecontroleerd of de waarde van antwoord gelijk aan acht is. De introductie van de '=' en "==" symbolen zorgen voor veel veranderingen in de syntax highlighter. Voor deze aanpassingen is de check op "is" in alle toestanden vervangen door de '=' symbool. Vergelijkbaar is de "is input[" vervangen door "= input[".

4.21 level 21

Tijdens conditie controles wil je ook kunnen controleren of een waarde niet gelijk is aan een ander. Om dit mogelijk te maken is er in dit level de "!" operand geïntroduceerd. Dit symbool betekent niet. Dit symbool wordt vaak gebruikt in combinatie met de vorig geïntroduceerde '=' teken. Een voorbeeld is te zien in figuur 29. In dit voorbeeld worden alle waarden behalve 5 geaccepteerd.

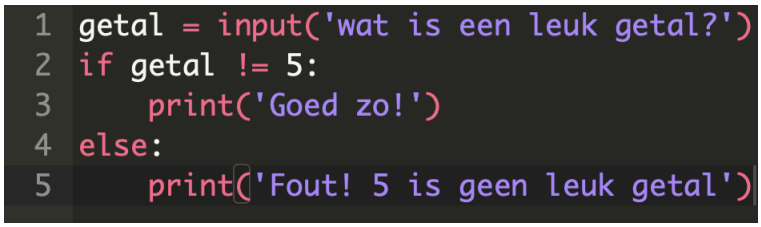

Figure 29: gebruik van '=' en "==" met de uitvoer rechts

Voor de implementatie van dit symbool in de syntax highlighter heeft er een kleine wijziging plaatsgevonden. In de "ifElseSpace" toestand wordt er nu niet alleen op de '=' gecontroleerd, maar vanaf nu ook op de "!=" symbool. Het is alleen in deze toestand geïmplementeerd, omdat de '!' niet in een andere commando gebruikt kan worden.

4.22 level 22

Level 22 is het laatste level in Hedy op dit moment. In dit level zijn de symbolen ">=" en "<=" geïntroduceerd. Het ">=" symbool controleert of een variabele of een waarde links van de groter of gelijk aan symbool groter is dan de waarde of variabele rechts van het symbool. En het "<=" symbool controleert of een variabele of waarde kleiner of gelijk is dan de waarde of variabele rechts van het symbool .Van beide symbolen is een voorbeeld beschikbaar. Zie hiervoor figuur 30 en 31. Voor deze functionaliteiten was er geen implementatie nodig in de syntax highlighter. Dit is, omdat

```
leeftijd = input('Hoe oud ben jij?')
if leeftijd >= 12:
    print('Dan ben je ouder dan ik!')
```

Figure 30: >= voorbeeld

```
leeftijd = input('Hoe oud ben jij?')
if leeftijd <= 12:
    print('Dan ben je jonger dan ik!')
```

Figure 31: <= voorbeeld

er binnen de "ifElseSpace" toestand zowel op de '>', '<' en '=' gecontroleerd wordt. Hierdoor is er geen aanpassing nodig in de higlighter.

5 moeilijkheden

Tijdens het implementeren van de syntax highlighter ben ik een aantal moeilijkheden tegengekomen, maar in het algemeen verliep de implementatie goed. Hieronder staan de moeilijkheden die ik ben tegengekomen.

- De documentatie van de syntax highlighter was niet helder genoeg. Er is voorbeeldcode aanwezig, maar het is niet altijd duidelijk waar dit specifiek toegepast moet worden. Na een tijdje ben ik erachter gekomen dat dit allemaal in een enkele bestand moest.
- Na het creëren van het bestand was het verbinden van dit bestand met Hedy niet helemaal duidelijk. Dankzij de hulp van een van de developers heb ik dit succesvol kunnen verbinden.
- Tijdens de development van de syntax highlighter is ervoor gekozen om een python module te gebruiken die niet ondersteund wordt op Windows. Dit probleem was echter niet gelijk duidelijk, want ik was één van de weinige die code schreef via Windows. Als oplossing op dit probleem, ben ik verder gaan ontwikkelen met een Macbook. Het was niet de moeite waard om een vergelijkbare Python module te vinden die Windows ondersteunde begreep ik van het develop team.

6 Conclusie

Het implementeren van een syntax highlighter is een leerzaam, maar ook een uitdagend process. Om syntax highlighting correct te implementeren moet je aan elke mogelijke combinatie van commando's en operanden denken. Hierbij moet je denken aan de transities die er plaats kunnen vinden en of bepaalde transities negatief uit kunnen pakken. Tenslotte moet er veel getest worden om zeker te weten of elke randgeval mee is genomen.

7 toekomstige onderzoek

De syntax highlighter kan in de toekomst verder uitgebreid worden met ondersteuning voor nieuwe levels die geïntroduceerd gaan worden. In deze levels kunnen er nog nieuwe commando's geïmplementeerd worden. De switch commando is hier een voorbeeld van. Daarnaast zijn er misschien randgevallen die nog niet verwerkt zijn in de highlighter. Om die randgevallen te vinden, moet er uitgebreid getest worden. Helaas was dat dit jaar niet mogelijk vanwege Covid-19.

References

[1] G. Dimitri. PPIG 2015: The impact of Syntax Highlighting in Sonic Pi. *PPIG*, 2015.

[2] T. Lyles. Apple's free learn-to-code swift playgrounds sandbox arrives on mac.

[3] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch. *Communications of the ACM*, 52(11):60–67, 2009.

[4] A. Sarkar. The impact of syntax colouring on program comprehension. *ACADEMIA*, 2015.

[5] I. Vale and A. Barbosa. Mathematical problems: the advantages of visual strategies . *Journal of the European Teacher Education Network*, 2018, 2018.

[6] J. Visser. Deze onderzoeker vindt dat ieder kind moet programmeren, 10 2017.

[7] Wikipedia-bijdragers". Reguliere expressie, 01 2020.