|  |
| --- |
| Paradigma challenge |

A pink rectangle on a black background

Description automatically generated

Inhoudsopgave

[1 inleiding 3](#_Toc210381977)

[2 Onderzoek 4](#_Toc210381978)

[3 Challenge 6](#_Toc210381979)

[4 Implementatie 7](#_Toc210381980)

[4.1 Importeren en exporteren 7](#_Toc210381981)

[4.2 htmlToMarkdown 7](#_Toc210381982)

[4.3 replaceHTML 7](#_Toc210381983)

[4.4 IsPrefixOf 8](#_Toc210381984)

[4.5 tags 8](#_Toc210381985)

[4.6 toLeft en geenSpatie 8](#_Toc210381986)

[5 reflectie 9](#_Toc210381987)

[6 COnclusie 10](#_Toc210381988)

[7 bronnen 11](#_Toc210381989)

[8 Conversaties 12](#_Toc210381990)

# inleiding

In dit verslag leg ik uit hoe ik de “Paradigma challenge” heb aangepakt. Het doel van deze challenge is om een programma te schrijven met een ander paradigma. In dit geval gaat het over functioneel programmeren. Hier moeten we een nieuwe taal voor leren en een programma kiezen om te bouwen

Ik heb voor de programmeertaal Haskell gekozen. Dit is een strikt functionele programeer taal. Er zitten dus helemaal geen objecten in. Een ander ding wat gelijk duidelijk wordt is dat er geen loops in deze taal zitten. Hier valt met recursie omheen te werken en daarom zie je ook veel recursie in mijn code. Door dit soort dingen word je geforceerd om functioneel te programmeren. Dit is iets wat ik erg interessant vind maar wat wel even wennen is.

# Onderzoek

Ik heb grotendeels van mijn onderzoek gedaan door te kijken naar een video serie

Deze video serie is gemaakt door een docent aan de universiteit van Nottingham. De Course is gemaakt voor de studenten daar in de klas en het matchte het niveau van programmeren waar ik opzat erg goed. Aan het eind van zijn lessenlaat hij een paar opdrachten zien. Deze kan je zelf maken om te kijken of je het goed begrijpt. Hierna gaat hij door de vragen heen en zo kan je dus zien of je het goed hebt begrepen of dat er nog wat dingentjes zijn waar nog aan gewerkt moest worden. Als ik iets verkeerd had ging ik altijd even onderzoek doen hoe ik dit beter kon doen en waarom. Ik heb veel van deze opdrachten niet meer. Maar een paar van mijn opdrachten en wat oefeningen die ik zelf bedacht heb staan in het bestand genaamd “test.hs” Hier heb ik allemaal losse functies die ik los oproep met ghci. Een van de dingen die ik erg interessant vond is de functie “nohoor”. Hier geef ik aan twee parameters nodig te hebben een variabele en een array van dezelfde soort variabele. Alles wat het doet is tellen hoeveel variabele in de array er kleiner zijn dan de eerste parameter. Dit had ik gemaakt om te gebruiken met cijfers. Maar ik kwam er al snel achter dat dit ook werkte met letters en woorden. Iets wat ik niet verwacht had en in een taal zoals Java veel moeilijker zou zijn. Dingen die ik heb geleerd zijn bijvoorbeeld

* **Higher-order functions**  
  Een functie die een andere functie als parameter meekrijgt
* **Recursie**  
  Een functie die zichzelf aanroept
* **Pattern matching**  
  Kijken welke conditie klopt en gebaseerd daarop de juiste code uitvoeren
* **Pure functie**  
  Een functie die geen zij effecten heeft
* **Immutability**  
  Het feit dat je variabele niet meer kan aanpassen nadat je ze hebt gemaakt
* **Lazy evaluation**  
  Onderdelen worden alleen gebruikt wanneer deze nodig zijn
* **Helper functie**  
  Een functie die word aangeroepen in een andere functie om ergens mee te helpen en de code overzichtelijk te houden
* **Currying**

Een voor een een argument meegeven aan je functie

# Challenge

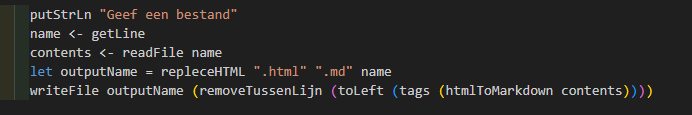
De challenge die ik heb gedaan is een HTML naar Markdown converter. Ik vind deze opdracht leuk omdat het een tool is die ik mogelijk ik de toekomst nog kan gebruiken. Het ingewikkelde zit vooral in het vervangen van tekst en hoe je dit het beste kan doen. Hiernaast is het herformateren van het bestand ook een uitdaging. HTML maakt veel meer gebruik van indentatie verschillen die erbij Markdown bijna niet zijn. Deze moeten weggehaald worden. Hiernaast pakken HTML en Markdown het begin en einde van een element heel anders aan. In HTML heb je een begin en een eindpunt. Zo heb je <div> en de div eindigt met </div> in Markdown bestaat dit niet. Hier wordt dit beslist per regel. Dus als ik een # gebruik dan werkt dat tot ik op enter druk.

# Implementatie

## Importeren en exporteren

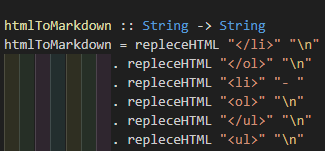
De implementatie geen best wel goed. Voordat ik begon wist ik ongeveer welke onderdelen ik nodig zou hebben. Dit zijn: een manier om HTML-bestanden uit te lezen, een manier om Markdown bestanden te exporteren, een loop die bij elk woord checkt of dit HTML-syntax is of een gewoon woord, een manier om witregels weg te halen en een manier om de verschillen in indentatie op te lossen.

Het inporteren heb ik gedaan door gebruik te maken van het getLine om te vragen naar de naam van het bestand en readFile om het bestand uit te lezen. Als alles gedaan is moet het bestand ook weer geporteerd worden. Dit doe ik door de bestandsnaam te gebruiken en .html te veranderen naar .md.  
Hierna kan ik het bestand exporteren met writeFile



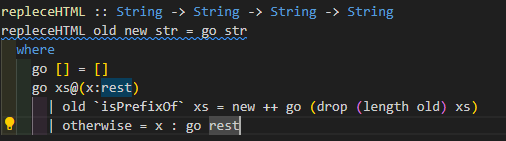
## htmlToMarkdown

Een groot deel van mijn code is het vervangen van HTML-tags met Markdown tags hiervoor heb ik de functie htmlToMarkdown die elke keer kijkt of hij er iets is om te vervangen. Ik had dit ook met een lijst kunnen doen waar doorheen gefilterd wordt in plaats van zo veel replaceHTML functies. Dan is het ook makkelijker om dingen toe te voegen en weg te halen



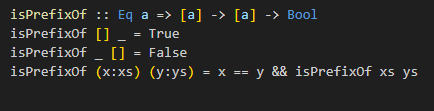
## replaceHTML

Het vervangen zelf wordt geregeld door replaceHTML. Dit is een recursieve functie waar er door een string wordt gekeken op zoek naar onderdelen om te vervangen. Hier maak ik gebruik van xs@(x:rest) zodat ik de array in zijn geheel kan oproepen maar ook in onderdelen. Hierna checkt hij of het begin van de string gelijk staat aan hetgeen wat vervangen moet worden. Als dit het geval is wordt deze vervangen en anders wordt er veder gegaan met de rest van de string.



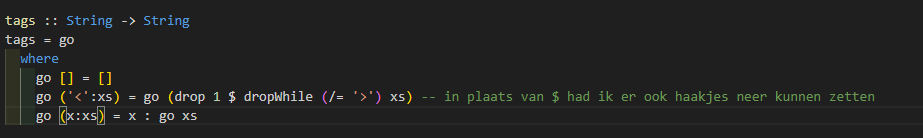
## IsPrefixOf

Zoals je bij replaceHTML zag werd is prefix of aangeroepen. Dit is hoe er gecheckt wordt of het iets is wat vervangen moet worden. Dit doet hij door elementen te vergelijken om te kijken of ze hetzelfde zijn tot de eerste lijst leeg is. Als alle elementen dan hetzelfde zijn dan geeft het true terug zo niet dan geeft de functie false terug. Dit wordt gebruikt om



## tags

Tags zorgt ervoor dat alles tussen de <> weg word gehaald. In HTML kan je bijvoorbeeld alt meegeven of dingen meer namen geven. Dit kan ik Markdown allemaal niet. De tags functie haalt dit allemaal weg. Dit gebeurt ook weer door een recursieve functie die alles dropt tussen de pijltjes.



## toLeft en geenSpatie

To left is een functie die lines verwijderd als geenSpatie true is.

GeenSpatie is true wanneer een zin of leeg is of als er /n staat

Dit is belangrijk omdat er veel lege lines zijn als je alleen de HTML-tags vervangt omdat alle </> tags weg zijn. Dit zorgt ervoor dat deze liens weg worden gemapt

# reflectie

Terugkijkend op dit project zijn er een paar dingen die ik best wel goed heb aangepakt. Normaal heb ik veel moeite met op tijd met opdrachten beginnen maar deze keer had ik besloten om zo snel mogelijk te beginnen met leren. Hierdoor had ik al vrij vroeg een redelijke basiskennis van Haskell en dat heeft mij erg veel geholpen. Ik had geluk dat al mijn kennis uit één video serie kon halen in plaats van rondzoeken om iets te vinden. Dit hielp ook erg met structuur omdat het direct duidelijk was hoeveel ik had gedaan en hoeveel ik nog moest. Het heeft voor mij vooral lang geduurd om comfortabel te worden mat een taal die zo anders is. Recursie goed gebruiken was voor mij even een struikelpunt. Dit was niet omdat ik het idee van recursie heel moeilijk vond maar vooral omdat ik een paar keer goed moest toepassen voordat het voor mij klikte hoe ik dit goed kan gebruiken. Er zijn ook wat dingen die ik niet helemaal goed heb aangepakt. Zo had ik zoals aangegeven bji htmlToMarkdown daar beter een lijst of iets kunnen gebruiken om daar doorheen te loopen.

# COnclusie

Dit was een erg leuk project waar ik veel van geleerd heb. Functioneel programmeren is iets wat ik erg interessant ben taan vinden en iets wat ik waarschijnlijk wel meer ga doen als ik de mogelijkheid krijg. Ik heb een kijkje gekregen in de wereld en de nadenk patronen van het functioneel programmeren. Het is een leuke andere manier van denken wat het voor mij een leuke en interessante uitdaging maakt.

# bronnen

*Functional Programming in Haskell*. (z.d.). YouTube. <https://youtube.com/playlist?list=PLF1Z-APd9zK7usPMx3LGMZEHrECUGodd3&si=POJ5v0d791UgGaiE>

*pure - HaskellWiki*. (z.d.). https://wiki.haskell.org/Pure

# Conversaties

https://<https://chatgpt.com/share/68de95f4-8388-8005-ab4a-8ed15562d68c>  
hier heb ik chat gebruikt om mijn logica te gebruiken op alle html bestanden en ze te vertalen naar Markdown

A black text on a white background

Description automatically generated