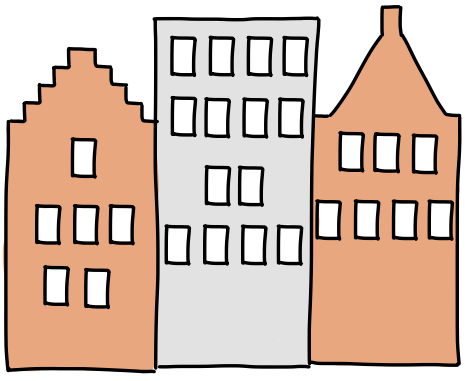
Sophie Straat, feministisch icoon en duider van deze tijd beschrijft in haar muziek de impact van software…

Je hoeft niet meer naar Amsterdam,

Het staat toch al op Instagram!

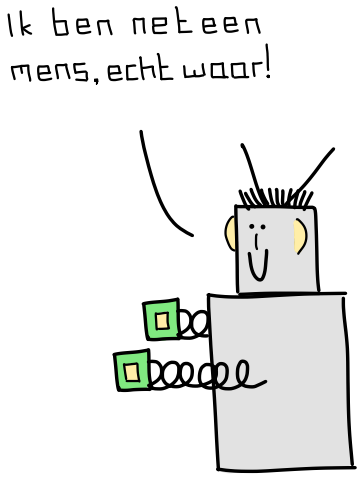


Amsterdam, Nederland en de wereld zijn ingrijpend veranderd door software. Kleine cafés worden overspoeld door Instagrammers, winkels zijn opeens flitsdepots, en huizen AirBnBs.

Als we als burgers, als mensen mee willen denken over oplossingen, hebben we kennis van software nodig.

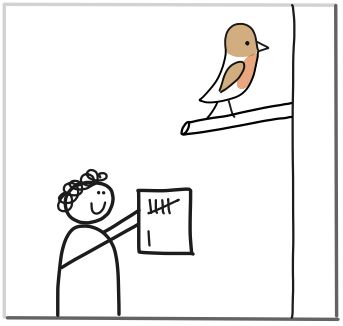
Stel je voor dat Halsema een eigen AirBnB wil laten maken. Kan dat? Kost dat 1000 euro of een miljoen? Hoe lang duurt dat? We kunnen alleen meedenken aan oplossingen als we snappen hoe software in elkaar zit.

Met nieuwe ontwikkelingen als GPT, die misschien nog meer impact hebben op de wereld om ons heen, is dat belangrijker dan ooit!



Zeker als de technologie steeds meer op mensen gaat proberen te lijken.

En het gaat niet alleen om het begrijpen van de digitale wereld, maar ook om het zelf iets erin kunnen maken!



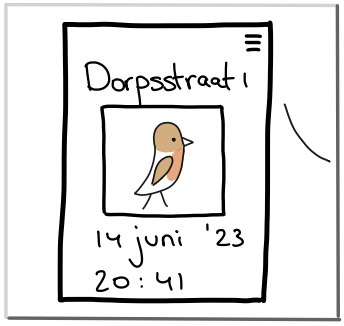
Zonder programmeren

Tellen

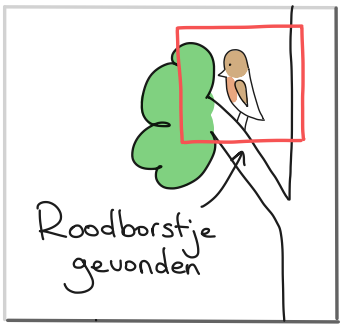
Stel je voor dat je bezorgd bent over het aantal roodborstjes in Nederland. Misschien omdat je fan bent of omdat het je werk is. Weet je niks van digitale zaken, dan kan je alleen maar roodborstjes tellen in je tuin.
Heb je veel sociaal kapitaal, dan tellen er misschien nog wat mensen met je mee.

Kan je een beetje programmeren?
Dan kan je denken aan een app, of beeldherkenning of tekstherkenning.
Je hoeft het niet eens zelf te maken, je kan het aan iemand vragen, of aan GPT!
Maar als je niet weet dat het kan, kan je het ook niet bedenken.

Met programmeren



App



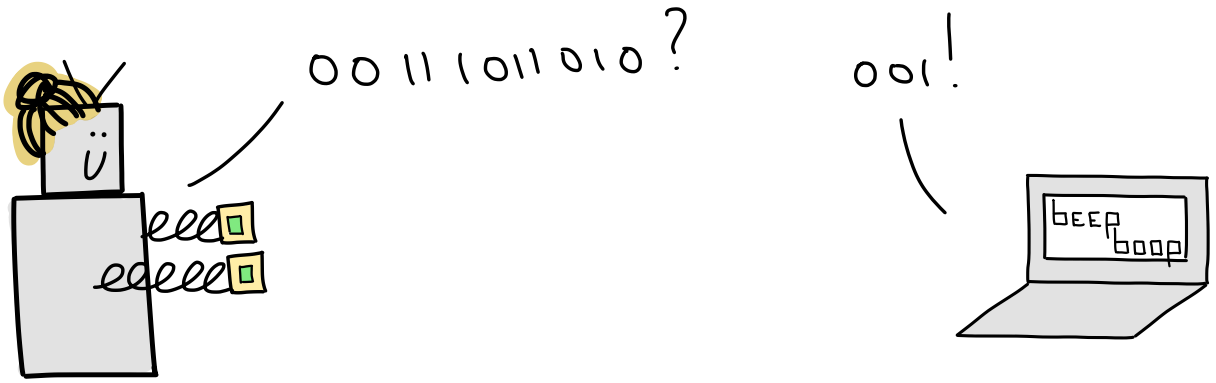
Beeldherkenning


HILVERSUM (GPD) - In de zesde aflevering van de documentaire reeks 'Vogels op woensdag' is vandaag het roodborstje de hoofdpersoon. Incest, moord en doodslag en fervente gevechten om het grondgebied zijn misschien niet direct begrippen die aan het roodborstje doen denken. Het leven blijkt echter ook voor dit vogeltje hard te zijn. De BBC filmde het gedrag van het roodborstje in de schilderachtige tuinen van Cranborne Manor in Dorset in Engeland. Nico de Haan van de Vogelbescherming geeft het commen-

Tekstherkenning

Daarom wil ik dat ieder kind, ieder mens, een beetje leert programmeren:
om toe te passen in ieder vakgebied, om de wereld om je heen net dat kleine beetje beter te maken, en om goede besluiten te kunnen nemen over het gebruik van technologie voor jezelf en voor de wereld.

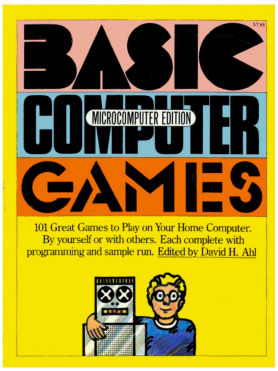
Maar als je gaat programmeren, dan moet je, als het ware, je menselijkheid achterlaten bij de deur. We moeten onszelf eigenlijk bijna in een computer veranderen om goed met computers te kunnen praten.



Maar dat zeg ik met de kennis van nu!
Toen ik ging lesgeven, zo'n tien jaar geleden, had ik geen idee!
Ik dacht eigenlijk dat het makkelijk zou zijn.

Zonder dat ik er echt bij stilstond, modelleerde ik de lessen op mijn eigen jeugd.
Dit ben ik toen ik een jaar of 10 was. Ik leerde mijzelf thuis programmeren, want we hadden een computer maar in die tijd zat daar helemaal niks op! Geen internet, geen videos, en geen spelletjes. Als je iets wilde doen met dat ding, moest je dat eerst zelf programmeren.

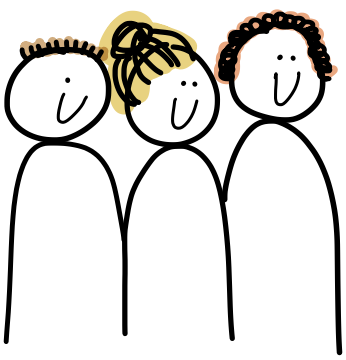
Ik leerde uit boekjes van de bieb…



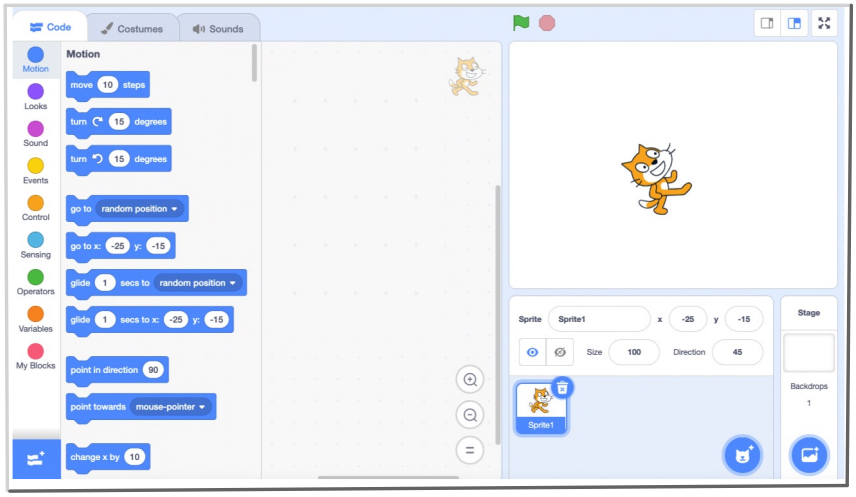
In die boekjes stond niet echt veel uitleg, alleen codes in het Engels.
Met veel overtypen en uitproberen, en heel veel tijd lukte het me wel om de codes te doorgronden.

```
135 INPUT "TIME INCREMENT (SEC)";S2
140 PRINT
150 INPUT "VELOCITY (FPS)";V
160 PRINT
170 INPUT "COEFFICIENT";C
180 PRINT
182 PRINT "FEET"
184 PRINT
186 S1=INT(70/(V/(16*S2)))
190 FOR I=1 TO S1
200 T(I)=V*C^(I-1)/16
210 NEXT I
220 FOR H=INT(-16*(V/32)^2+V^2/32+.5) TO 0 STEP -.5
221 IF INT(H)<>H THEN 225
222 PRINT H;
225 L=0
230 FOR I=1 TO S1
240 FOR T=0 TO T(I) STEP S2
245 L=L+S2
250 IF ABS(H-(.5*(-32)*T^2+V*C^(I-1)*T))>.25 THEN 270
260 PRINT TAB(L/S2);"0";
270 NEXT T
275 T=T(I+1)/2
276 IF -16*T^2+V*C^(I-1)*T<H THEN 290
280 NEXT I
290 PRINT
300 NEXT H
310 PRINT TAB(1);
320 FOR I=1 TO INT(L+1)/S2+1
330 PRINT ".";
340 NEXT I
350 PRINT
355 PRINT " 0";
360 FOR I=1 TO INT(L+.999S)
380 PRINT TAB(INT(I/S2));I;
```

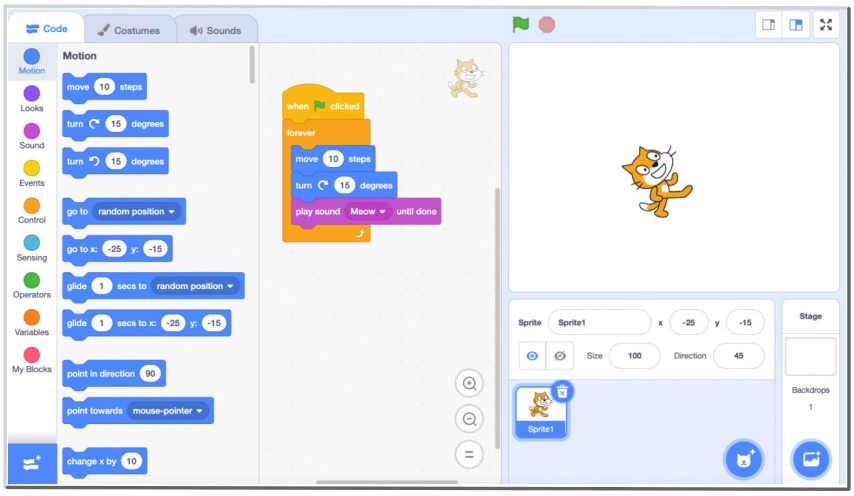
En ik ben niet de enige! Veel mensen van mijn leeftijd die nu programmeren, die kind waren in de jaren 80 en 90, hebben zichzelf leren programmeren.
Zonder lessen of leraar en met veel fouten maakten we het ons eigen.



Ik begon de lessen met Scratch, een blokkentaal, een soort Lego. Je zet er zo iets leuks mee in elkaar.

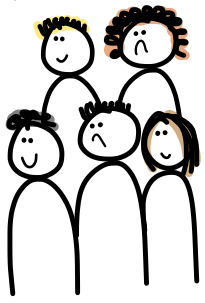


Met dit programma maak je bijvoorbeeld al een ronddraaiende en miauwende kat.

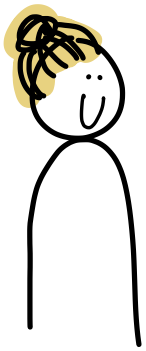


Dat was leuk maar, er was ook een probleem…

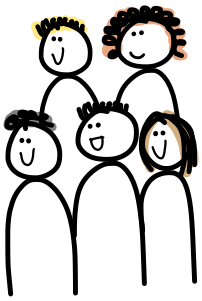
Dit is kinderachtig



Kinderen van een jaar of twaalf wilden geen speelgoedtaal meer, ze vonden zichzelf oud genoeg voor een grotemensentaal.



Geen probleem!
Laten we Python leren.



Maar in Python moet je heel veel tegelijk onthouden.
Codes, en haakjes en aanhalingstekens. Dat viel niet mee!

Met deze code zet je tekst op het scherm:

```
print("Hallo allemaal")
```

Hier komt de "onmenselijkheid van de computer" om de hoek kijken.
Maak je een foutje, dan loopt Python meteen vast.

Gebruik je per ongeluk een hoofdletter: Foutmelding.

```
Print("Hallo allemaal")
```

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    Print("Hallo allemaal")
NameError: name 'Print' is not defined
```

Vergeet je een haakje sluiten: Foutmelding.

```
print("Hallo allemaal"
```

```
  File "main.py", line 2
                        ^
SyntaxError: unexpected EOF while parsing
```

Per ongeluk een spatie getikt: Foutmelding.

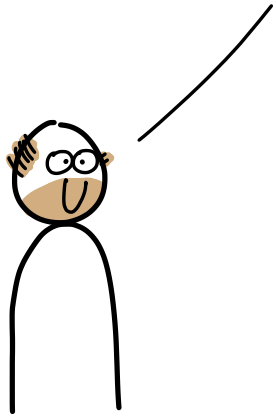
```
 print("Hallo allemaal")
```

spatie!

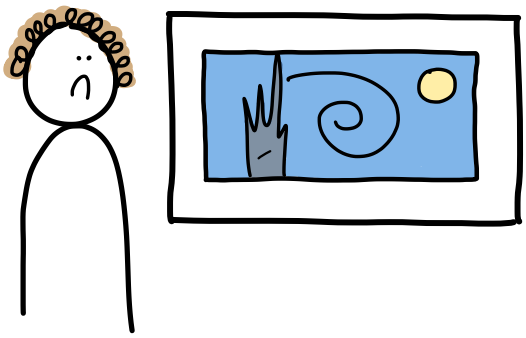
```
  File "main.py", line 1
    print("Hallo allemaal")
    ^
IndentationError: unexpected indent
```

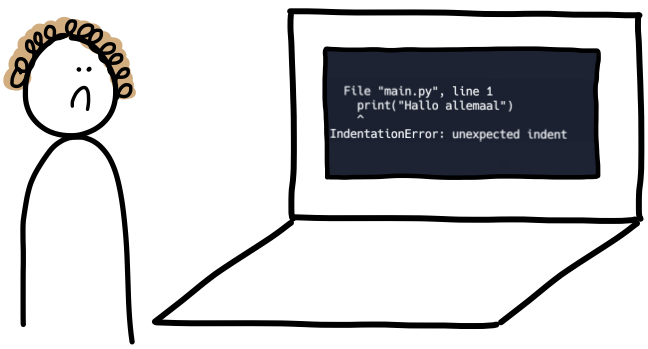
Mark Weiser, invloedrijke denker over technologie en CTO van Xerox PARC, zei ooit…

Een goede tool is een onzichtbare tool



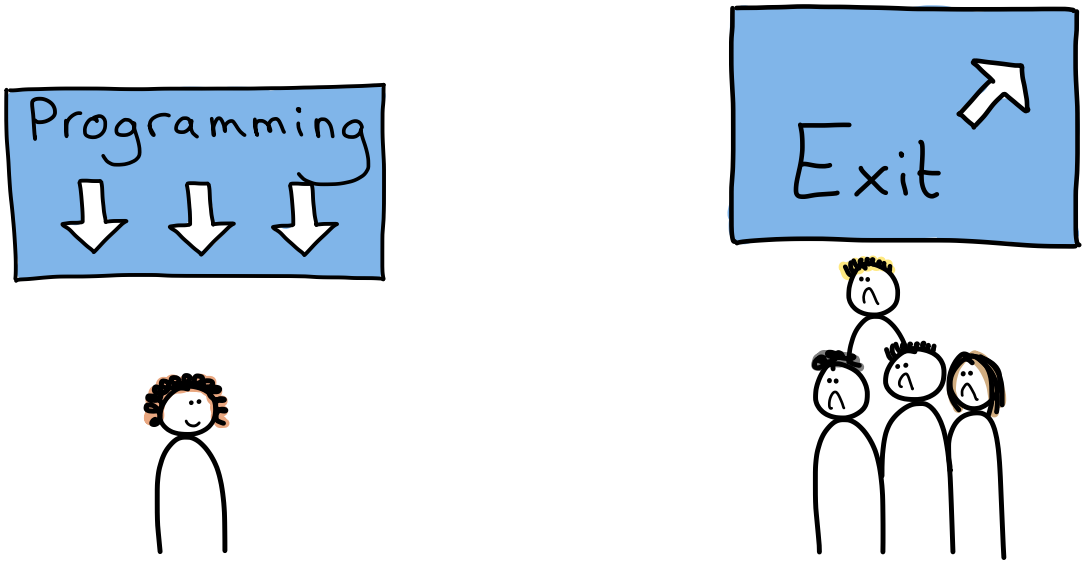
Je bent aan het breien, niet breien met een breipen, of schilderen met een penseel. Als je schildert, focus je je op het doek, niet op de penseel. Dat gebeurt alleen als er iets misgaat zoals wanneer er een haartje loskomt. Dan springt de tool opeens in beeld.





Zo is het ook met programmeren; een foutmelding maakt je opeens bewust van de programmeeromgeving.

De kinderen in mijn lessen waren het met al die fouten snel zat. De meesten vonden er al vlot geen bal aan, ze gingen lekker spelletjes spelen, of ze hielden er mee op. In ieder geval leerden ze niet echt wat, op een enkeling na.



De kinderen die afhaakten waren vaak de kinderen zonder veel programmeerervaring: kinderen zonder computer thuis, en kinderen met weinig zelfvertrouwen in hun vaardigheden, vaak meisjes. Ze vonden het al spannend en dachten dat het moeilijk zou zijn, en zagen in de foutmeldingen hun voorgevoel bevestigd.

Als we het hebben over weinig meisjes in de informatica, dan hebben we het vaak over rolmodellen of over discriminatie, maar nooit over het ontwerp van programmeertalen, terwijl foutmeldingen met een laag zelfvertrouwen heel anders voelen dan als je je al bijna programmeur voelt.

Toen werd ik bevangen door een nieuwsgierigheid naar leren leren! Hoe leren mensen eigenlijk dingen, welke dingen dan ook? De kinderen in mijn klas konden lezen en schrijven, dus ze konden prima leren. Ze moesten, in theorie, dus ook kunnen leren programmeren. Ik ging me op dat moment echt verdiepen in leren.

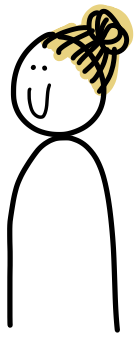
Gelukkig had ik toen een geweldige collega: Marileen Smit, vandaag ook aanwezig hier. Opgeleid als ontwikkelingspsycholoog, wist zij een hoop over leren en wat ze niet wist, zocht ze op. Ik herinner me deze tijd als een soort tweede studie, waarin ik iedere week domme vragen mocht stellen, en zij me geduldig vertelde welke boeken ik moest raadplegen en hoe ik naar de zaak moest kijken.

Ik ben haar enorm dankbaar voor haar geduld en tijd en kijk met grote warmte terug op onze samenwerking in die tijd. Marileen, dankjewel.

Ik leerde een hoop, maar in traditionele zin was mijn carrière aan het mislukken.
Ik had al tijden geen nieuwe dingen meer bedacht en geen papers meer geschreven.
Dat was geen makkelijke tijd.

Op goede dagen zei ik tegen mijzelf:

Geen zorgen!
Diep nadenken is
precies het werk van
een wetenschapper



Maar op slechte dagen dacht ik:

Ik ben een mislukking
want ik produceer niks

Na een hele lange fase van nadenken, kwam ik wel tot een paar zinnige inzichten, gebaseerd op hoe we andere dingen leren. Leren in andere vakgebieden gaat meestal in stapjes: je leert niet meteen nevenschikkende bijzinnen in groep 3.

Eerst maar eens letters…

dan woordjes…

dan zinnen.

Toen dacht ik: ik kan zelf een programmeertaal maken! Ik kan dus ook een taal maken die dit stramien volgt: eerst heel losjes, en dan steeds strenger, met meer regels. Ik ga dat bouwen en proberen!!

Dus na heel veel denken, en nu hebben we het dus echt over de periode van begin 2016 toen ik aan onderzoek naar leren programmeren begon, tot eind 2019, maakte ik Hedy: een programmeertaal gemodelleerd op hoe mensen leren.

Met Hedy kan je in kleine stapjes leren, je leert maar een paar codes tegelijk.
In het begin mag je van alles, en pas later komen er strengere regels bij.

In Hedy level 1 hoef je geen aanhalingstekens of haakjes te gebruiken.

```
1  print Hello!
2  print Welcome to Hedy!
3
```

Pas in level 4 worden aanhalingstekens verplicht.

```
1  print 'Hello!'
2  print 'Welcome to Hedy!'
```

Maak je een foutje?
Hedy legt in een zinnetje uit wat er mis is, in het Nederlands!

```
1  print 'Hello!'
2  print 'Welcome to Hedy!
```

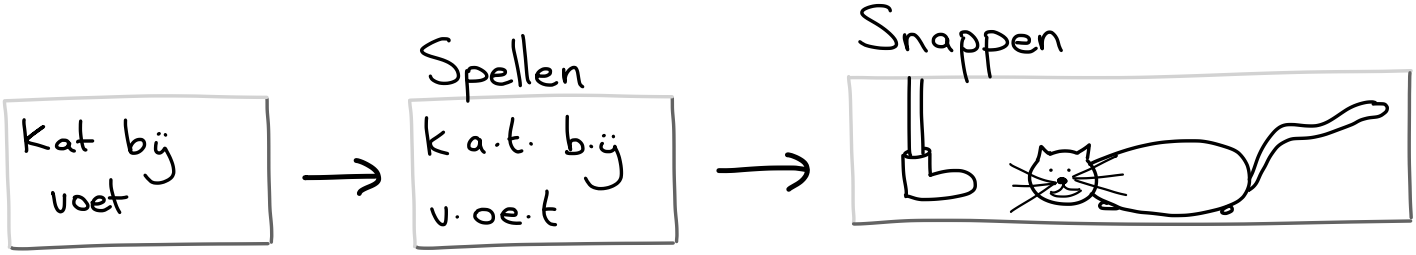
We konden je code niet goed lezen.

Let op! Bij `print` en `vraag` moet voor én achter de tekst een aanhalingsteken komen. Jij bent dat vergeten voor de tekst 'Welcome to Hedy!.

Nu zei ik net "en toen maakte ik een nieuwe programmeertaal Hedy" maar hoe maak je eigenlijk een programmeertaal?
Laten we daar even induiken.

Het meest simpele antwoord is: met een andere programmeertaal!
Er zijn speciale programmeertalen die je kan gebruiken om programmeertalen te maken. Dat is handig, dan hoef je niet alles helemaal zelf te doen.

Een programmeertaal maken doen we in twee stapjes: spellen en snappen.
Vergelijk het maar met een kind van 6 dat woordjes leest en er een tekening bij moet maken. Eerst ga je letter voor letter een woord spellen: k - a - t. Dan luister je wat je gespeld hebt en dan bedenk je of je weet wat dat woord betekent. Weet je wat dat woord is, dat kan je het tekenen.

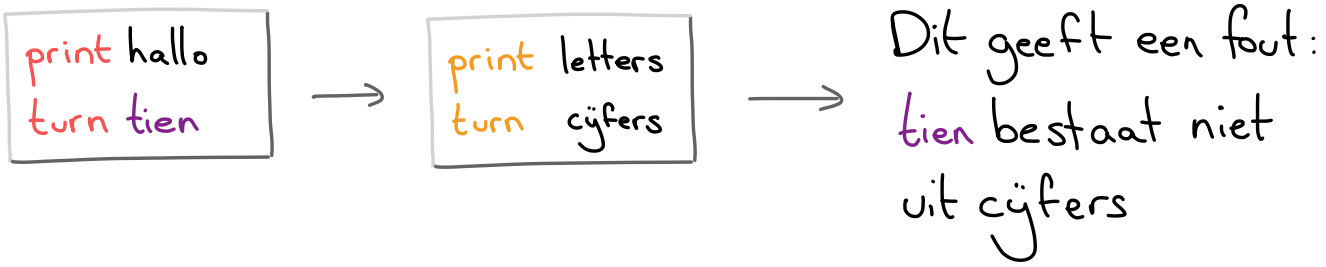


Bij de computer gaat het net zo. Ten eerste moet je aan de computer uitleggen welke letters er allemaal gelezen kunnen worden. De computer leren spellen als het ware.
Dat doen we met een grammatica.

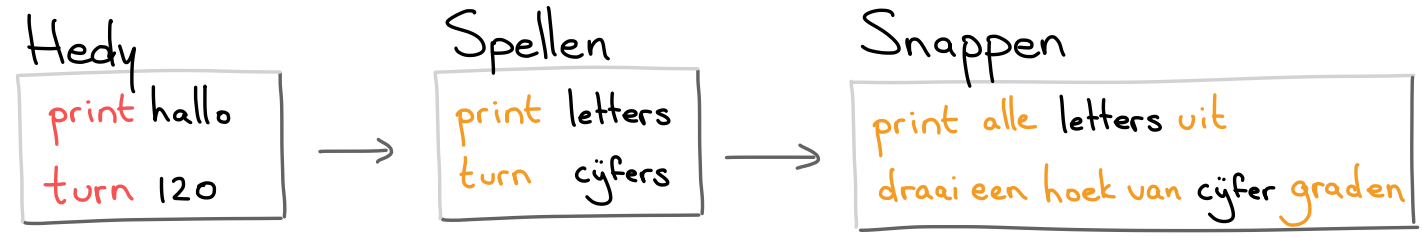
Dit is een stukje van de Hedy grammatica:

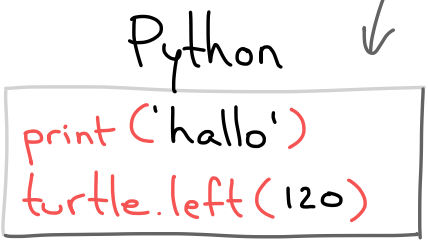
```
print letters
turn cijfers
```

De computer kijkt dan of een gegeven code klopt, dat noemen we parsen.



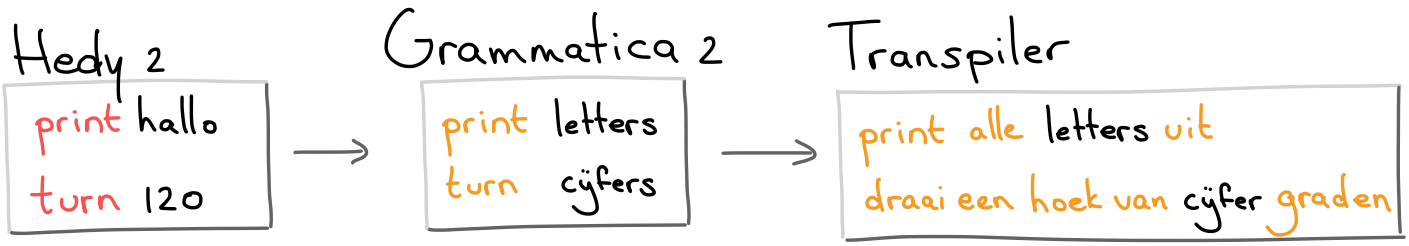
En dan het snappen. Misschien weet je dat een computer alleen “nulletjes en eentjes” kan lezen. Dus misschien denk je dat we dan in nullen en enen moeten vertellen wat print “hallo” betekent. Dat kan!

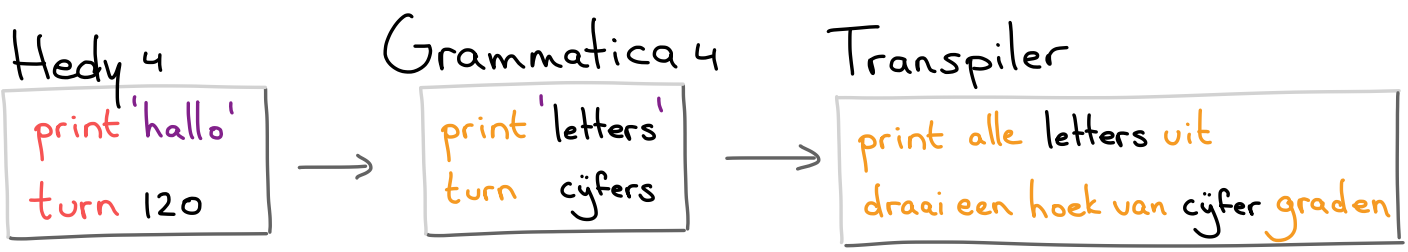




Maar dat hoeft niet. Omdat er al zoveel programmeertalen zijn, kun je een list gebruiken.
Je kan zeggen: deze Hedy code vertalen we naar deze Python code. En een computer weet al hoe hij Python moet snappen, dus dat is makkelijk!

Maar Hedy is anders dan andere talen, omdat het in stapjes werkt. En grammatica’s die je gebruikt om andere programmeertalen te maken kunnen geen stapjes aan. Omdat er nog nooit zo’n taal is geweest met stappen, kunnen de grammatica’s dat nog niet.

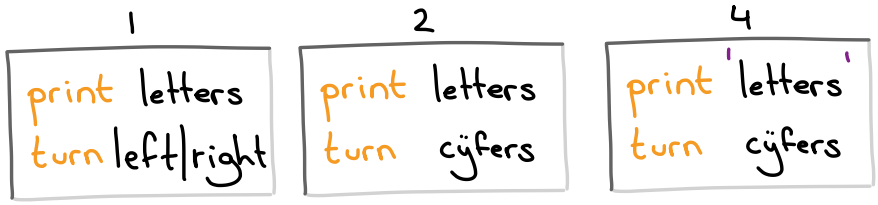




Er was dus extra werk nodig om Hedy te maken.

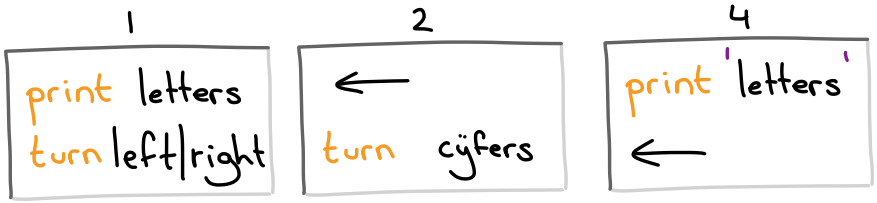
Versie 1

In eerste instantie maakte ik niet één taal, maar allemaal losse talen, met copy en paste. Dat werkte wel, maar als je dan iets moest aanpassen, moest dat in alle kopietjes. Niet zo handig als je vaak dingen wilt aanpassen naar aanleiding van feedback.

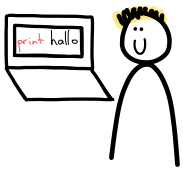


Versie 2

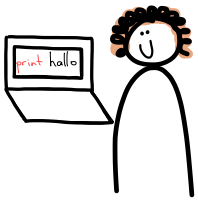
Dus toen heb ik een aanpassing gemaakt in de grammatica's, en nu kan je een soort "deeltalen" maken, en in het volgende level zeggen: doe mij maar wat we al hadden, met deze dingen erbij en die eraf.



In mijn onderzoek naar leren programmeren, doe ik dus twee dingen.

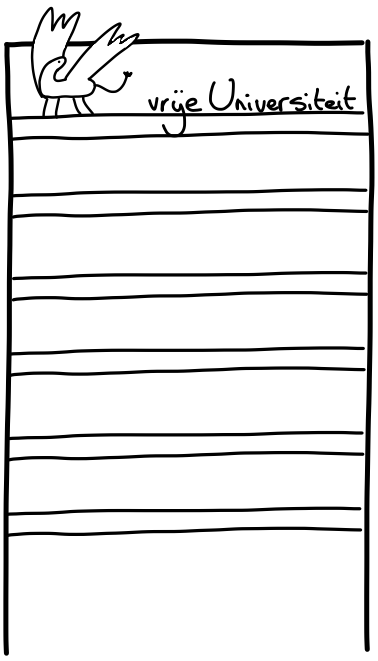


Enerzijds informaticaonderzoek:
nieuwe programmeertalen maken voor programmeertalen maken.
Leren programmeren is de aanleiding voor de vraag of we een stapjestaal kunnen maken.



Aan de andere kant is "kijken of kinderen met Hedy fijner leren programmeren" echt mensen-onderzoek.

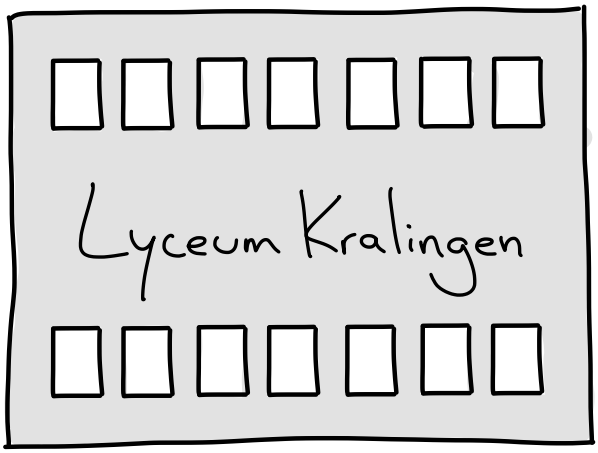
Daarom is het ook zo geweldig dat ik hier vandaag sta namens twee faculteiten:
Ik kan bij gedragswetenschappen onderzoek doen naar mensen, en mijn collega's daar snappen hoe moeilijk het is om een goede observatiestudie te doen,
én ik mag ook bij informatica computernerd zijn en puzzelen aan moeilijke programmeervraagstukken.



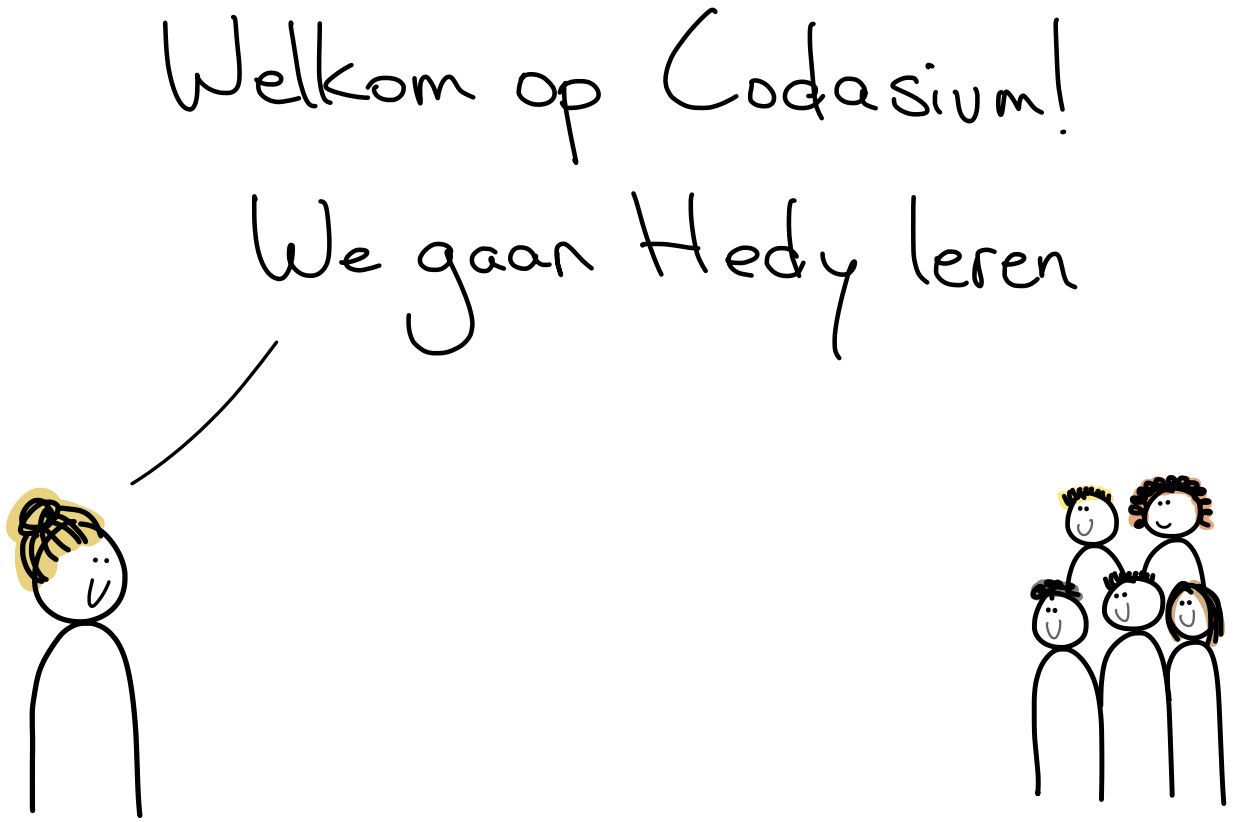
Ik mag mijn hele zelf zijn.

Naast mijn werk aan de VU mag ik dan ook nog op een school in de praktijk brengen wat ik aan beide kanten leer. Lyceum Kralingen is voor mij echt een tweede thuis, dankzij de geweldige leerlingen en fijne collega's.

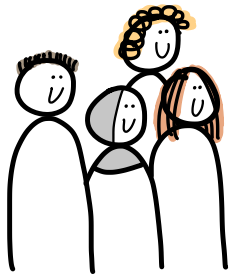
De vrijheid die ik daar altijd krijg om te blijven leren is ongekend!



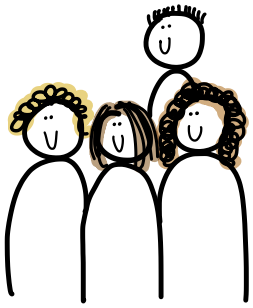
Dus Hedy een heel klein beetje stond, ging ik dat doen: leren van leerlingen op school.



En als je dan gaat kijken wat er gebeurt, dan leer je echt verrassende dingen!



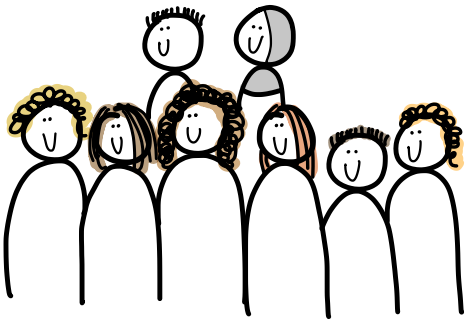
Hedy



Python

Vaak denken mensen dan meteen aan meten: becijferen of Hedy beter is dan bijvoorbeeld Python of Scratch.

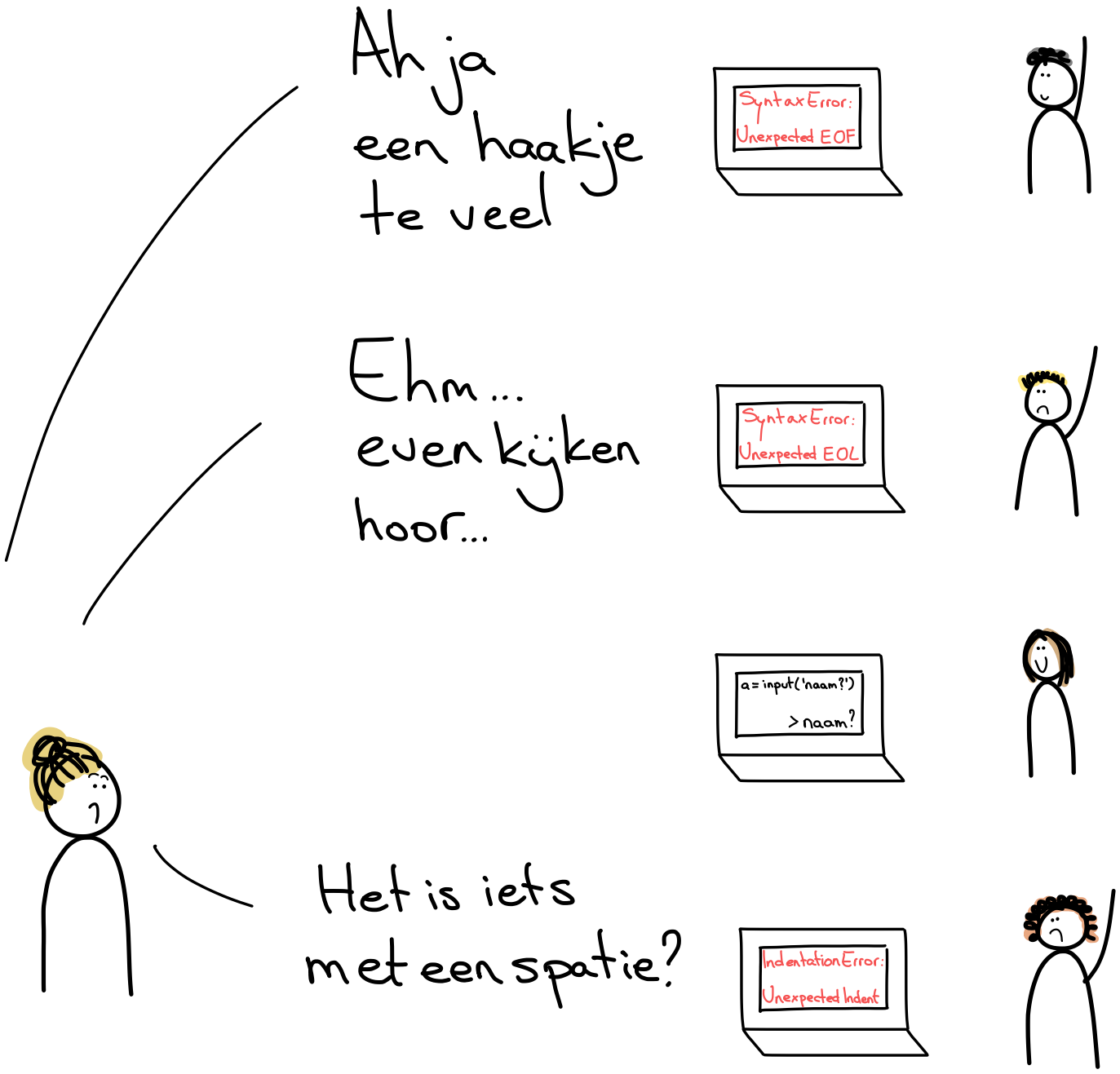
"Meten is weten" klinkt goed maar wat moet je dan eigenlijk meten? Cijfers? Enthousiasme? Begrip operationaliseren is niet alleen heel moeilijk, je weet vaak helemaal niet wat je wilt meten.



Hedy

Er is niks mis met meten, maar er zijn nog zoveel andere manieren om over de wereld te leren. Soms kom je al een heel eind met gewoon rustig rondkijken.

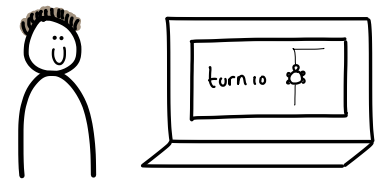
Bijvoorbeeld: Leerlingen die in Python een foutmelding kregen, konden daar eigenlijk nooit zelf mee uit de voeten. Als docent moest je altijd meehelpen, en moest je vaak zelf ook nog even puzzelen wat er mis was.



In een programmeerles loop je dus vaak achter de feiten aan, er zijn altijd meer handen in de lucht dan tijd om te helpen. Per fout ben je zo een paar minuten kwijt, want je wilt de fout oplossen en ook uitleggen waar de fout door kwam.

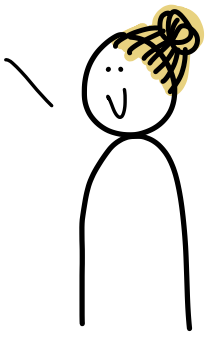
Dit is wat veel docenten ervan weerhoudt om met leerlingen te gaan programmeren, dat de klas snel chaotisch wordt.

Maar, met de foutmeldingen van Hedy konden leerlingen wel zelf vooruit. De docent kan vol vertrouwen zeggen: de foutmelding zegt waar de fout zit.



Kijk nog eens goed op regel 4!

Lees de foutmelding nog eens!



Het is dus niet zo dat Hedy per se minder foutmeldingen geeft, maar dat je minder tijd kwijt bent per fout, leerlingen dus beter kan helpen en er daardoor rust in de klas ontstaat.

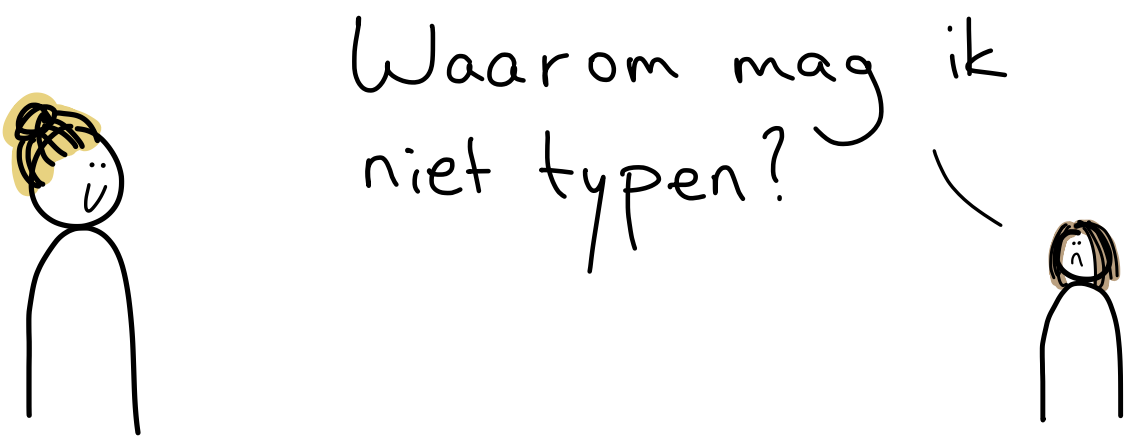
Pas als je eerst eens gaat kijken, kom je tot dingen die je misschien wel zou kunnen en willen meten.

Als je de tijd hebt om rustig te kijken zie je ook piepkleine dingen. Bijvoorbeeld deze foutmelding. Ik vond het duidelijk, maar leerlingen niet!

```
1  print Welcome, world!
```

We konden je code niet goed lezen.

De code die jij intypte is geen geldige Hedy code. Er zit een foutje op regel 1, op positie 14. Jij typte ',' maar dat mag op die plek niet.



Dit is nu de foutmelding!

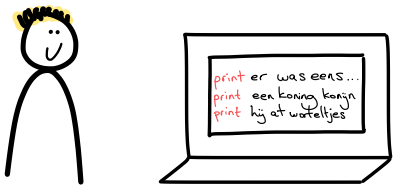
```
1  print Welcome, world!
```

We konden je code niet goed lezen.

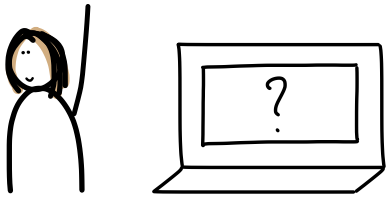
De code die jij intypte is geen geldige Hedy code. Er zit een foutje op regel 1, op positie 14. Jij typte een komma, maar dat mag op die plek niet.

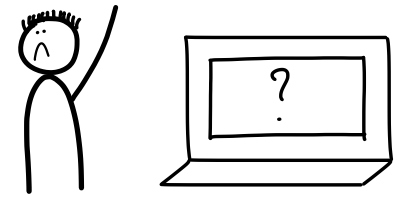
Zo schaafden we op school verder aan Hedy, ook andere docenten gingen ermee aan de slag. Met de verbeterde foutmeldingen en user interface, kwam er weer ruimte om na te denken over de kern van het probleem.

In andere klassen zagen we dat kinderen vaak niet wisten wat ze moesten maken.

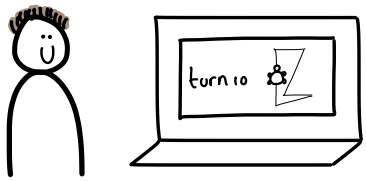


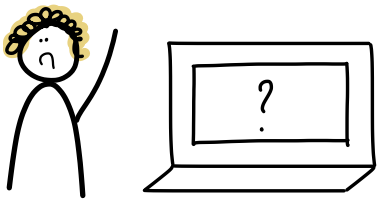
Voor veel leerkrachten was helpen moeilijk, omdat ze zelf niet wisten wat er allemaal kon. Sommige kinderen gingen dan lekker aan de slag, maar anderen helemaal niet!

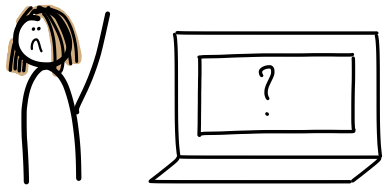


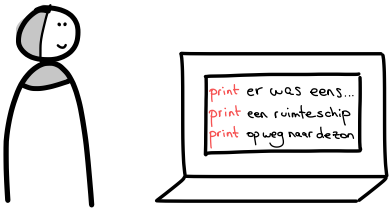


Ik had dat probleem zelf niet zo, omdat ik al veel over de mogelijkheden van Hedy had nagedacht, en omdat ik goed wist wat bij hun niveau zou passen. Dit zag ik pas bij andere docenten in de klas!

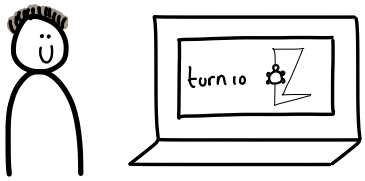




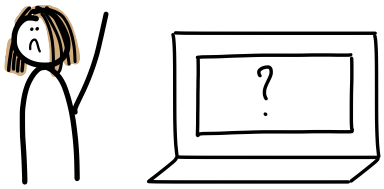


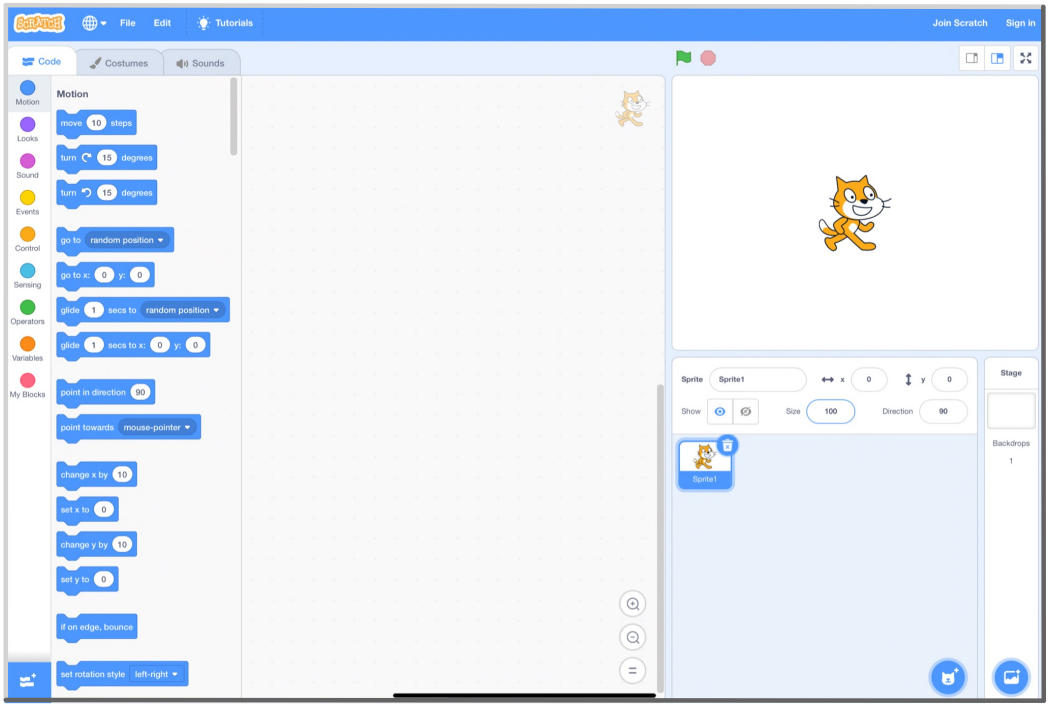


Nu denk je misschien: “ja jammer voor die kinderen die niet wisten waar ze moesten maken”, maar er zit meer achter. Het waren niet zomaar kinderen die vaak niet aan de slag gingen, het waren weer vaker meisjes! Dat was opvallend, vond ik, want meisjes zijn over het algemeen creatief! Wisten ze niks te verzinnen?

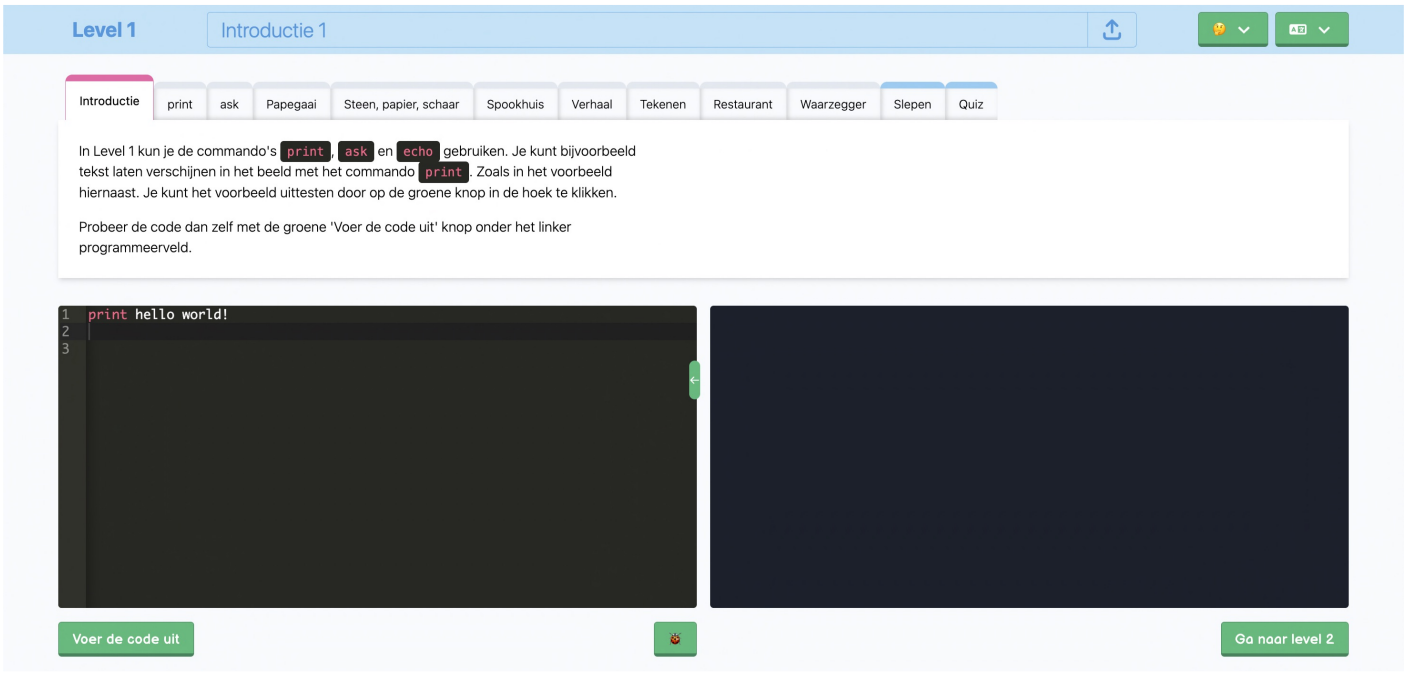


Over het algemeen vinden meisjes het niet zo leuk om te “*tinkeren*”, om uit te proberen. Liever verzamelen ze eerst informatie over wat er allemaal mogelijk is. En ze zijn vaker gemotiveerd door wát je kan maken, niet door het maken zelf.

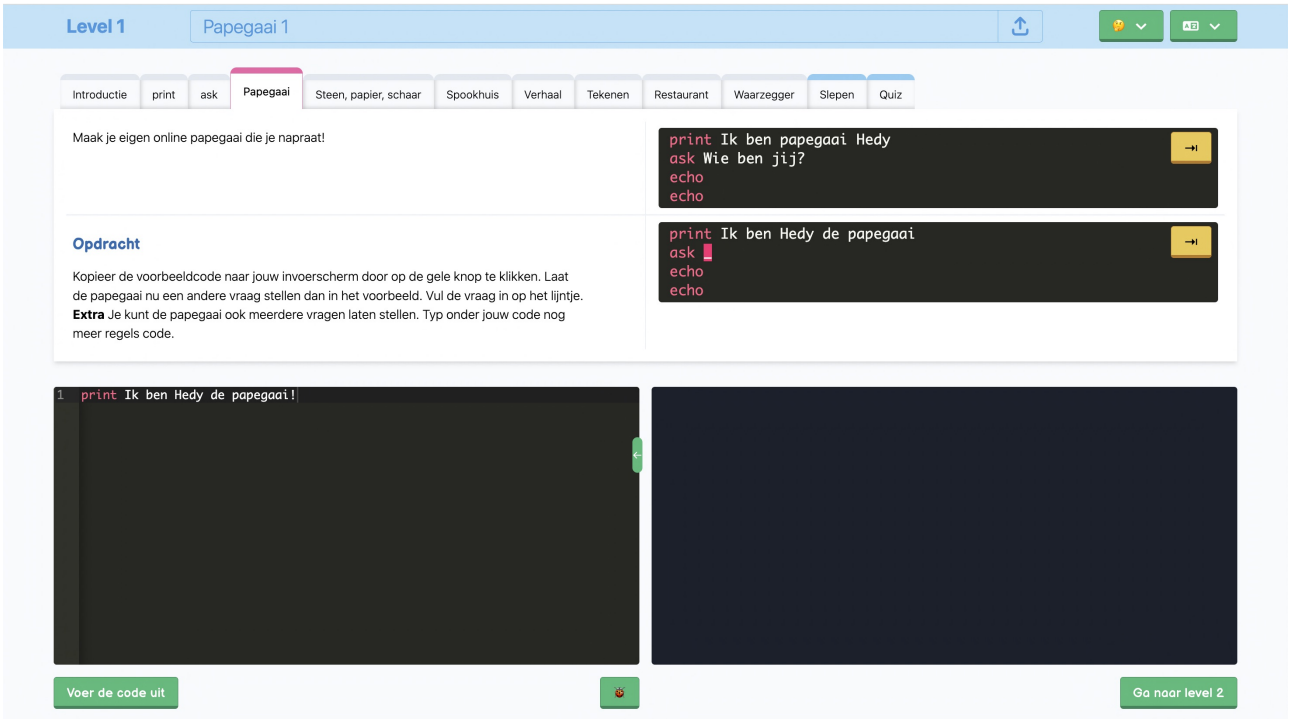




Als je met die wetenschap naar Scratch kijkt, dan zie je dat het genderneutraal lijkt: er staan immers geen typische jongensdingen op zoals een auto of voetbal. Maar het is toch heel jongensachtig:
er is nergens uitleg te vinden en het legt niet uit wat je ermee kan maken.



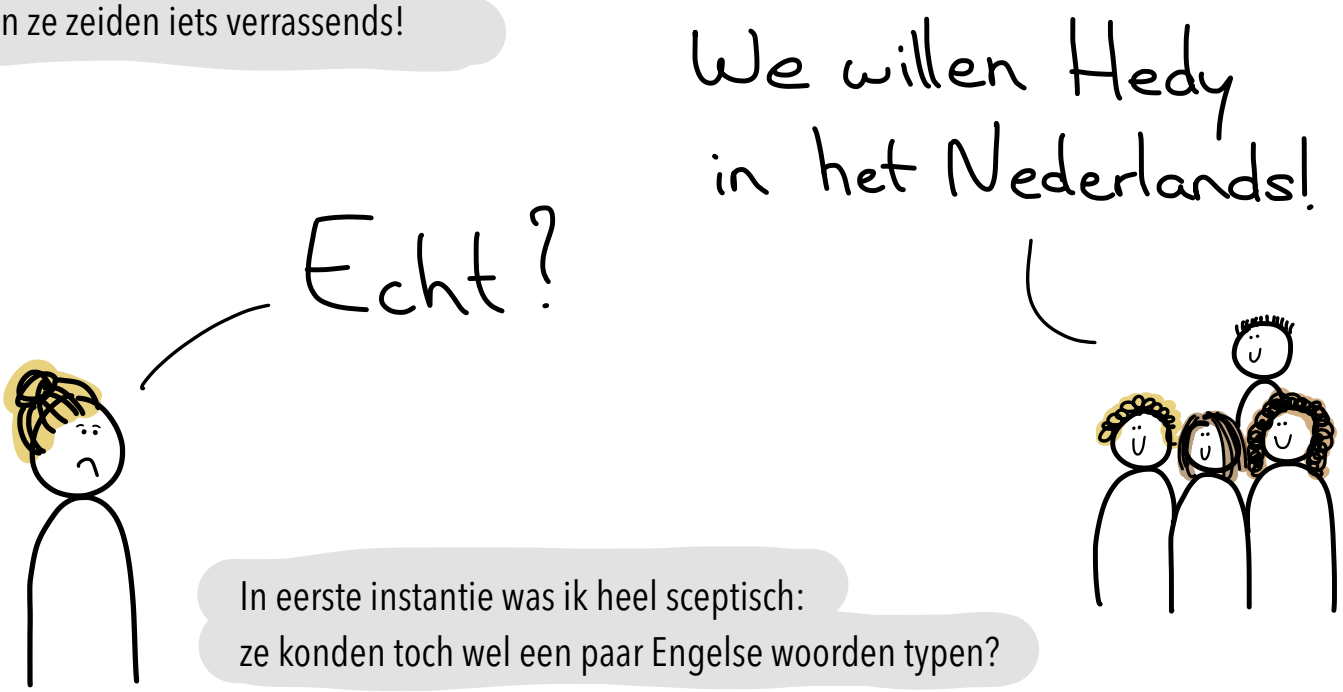
Hedy heeft daarom ingebouwde uitleg (wij noemen dat: een avontuur) die stap voor stap zegt wat je kan doen. Leerlingen en docenten vinden dat fijn, en wie wil experimenteren kan de uitleg overslaan.



De uitleg is niet zo'n duidelijke technische uitvinding als de graduele stapjes, maar misschien wel net zo belangrijk. Veel docenten zeggen dat dit Hedy zo makkelijk maakt: alle leerlingen kunnen er echt zelf mee uit de voeten. En als docenten willen, kunnen ze zelf hun eigen uitleg inladen.

Maar, ik keek niet alleen zelf! Ik vroeg ook aan leerlingen wat zij willen. Ik geloof echt in *participatory* onderzoek, waarin deelnemers niet alleen bekeken worden maar ook meedoen als onderzoekers en ontwerpers.

En ze zeiden iets verrassends!

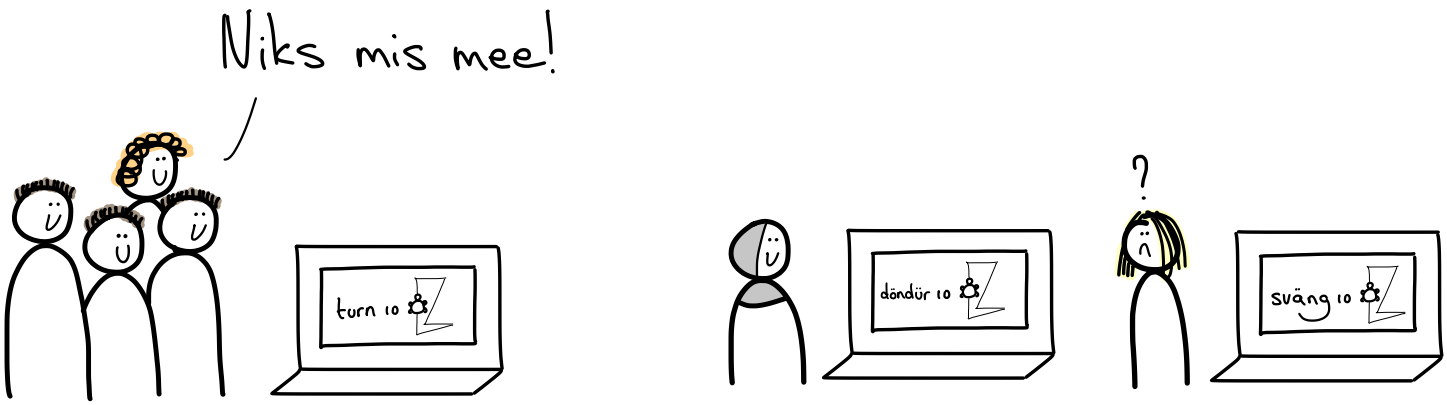


In eerste instantie was ik heel sceptisch: ze konden toch wel een paar Engelse woorden typen?

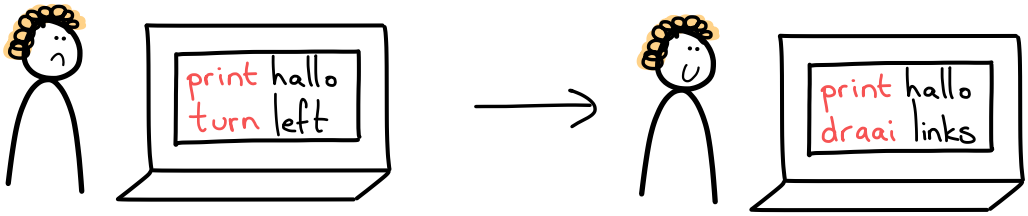
Maar de leerlingen zetten me wel aan het denken… waarom zijn talen in het Engels?

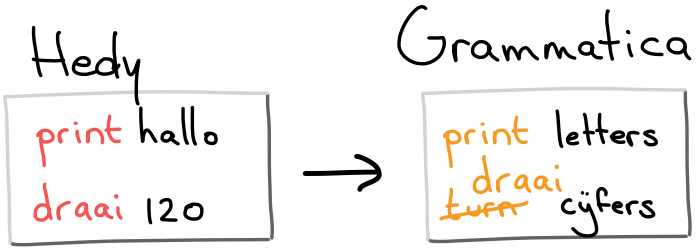
Een antwoord is dat het zo is omdat het is zo is. De eerste talen waren in het Engels, dat bleef zo, en iedereen die nu programmeertalen maakt, kan al Engels en zo blijft de status quo in stand.

Een andere reden is dat het niet handig is als Fatima in het Turks programmeert en Astrid in het Zweeds, want dan kunnen ze elkaars codes niet lezen. Daarom zijn sommige mensen fel tegen het idee.



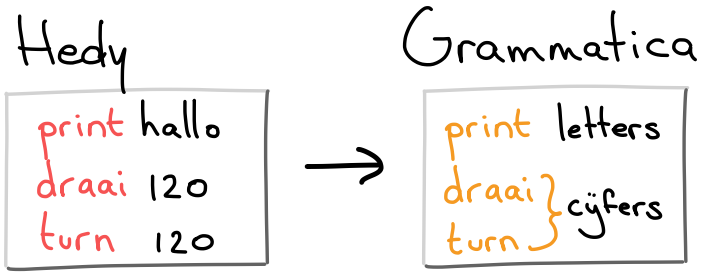
Ondanks mijn scepsis, vond de computernerd in mij het wel een leuke puzzel!



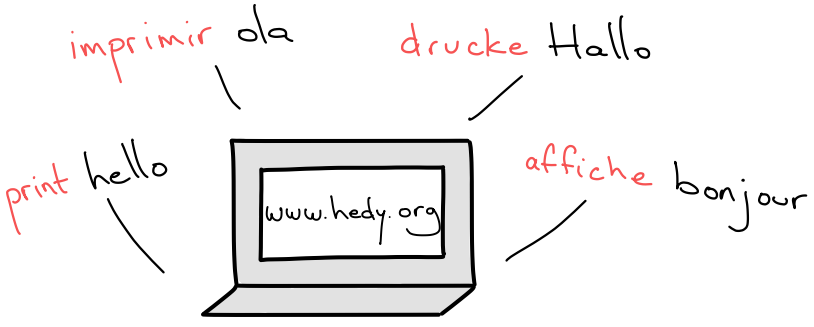


De eerste stap was de Engelse codes vervangen door Nederlandse.

Maar! Sommige kinderen kunnen al een beetje Engels! Dus nog beter is het als je talen mag mixen. Dat kan nu ook.

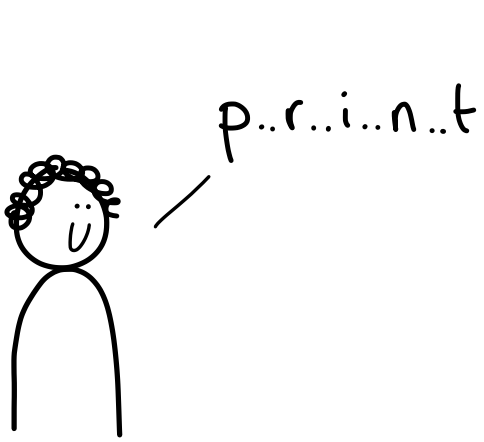


Ik was best trots op mijzelf want ik had een meertalige programmeertaal gemaakt, en na Nederlands kwam er al snel Frans bij en Duits en Spaans….

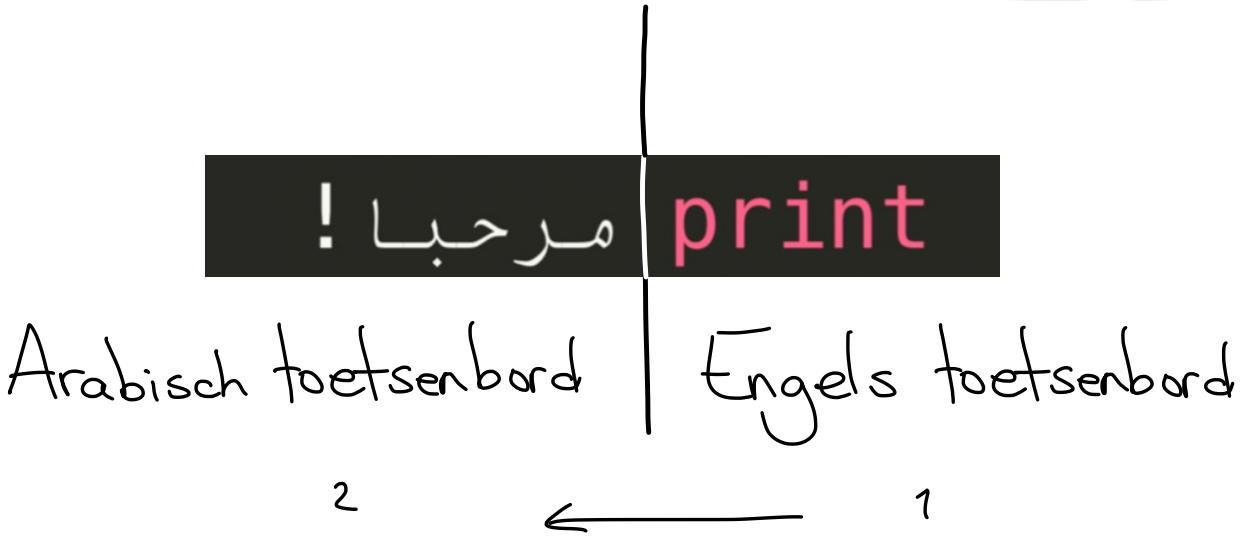


Maar toen zei mijn goede vriend Alaaeddin Swidan…

Stel je voor dat je een Arabische juf of meester bent en je moet print uitleggen.



Pas toen zag ik hoezeer de leerlingen gelijk hadden! Als je tegelijk programmeren en een nieuwe taal moet leren, is dat extra moeilijk. Je legt dan een hogere barrière op aan sommige kinderen, zeker als die taal ook nog met nieuwe letters is!



Maar een tweede reden is een ergonomische reden. Als je Arabisch en Engels door elkaar typt, dan moet je van softwarematig toetsenbord wisselen.
Je moet dan dus wisselen naar Engels, print typen, dan wisselen naar Arabisch, dan مرحبا typen en dan weer terug wisselen. Heel heel onhandig!

Technisch was het echter een ingewikkelde puzzel. Om goed te snappen waarom dat zo is, moeten we weer kijken hoe programmeertalen gemaakt worden.
In de basis is Arabisch niet moeilijker dan Engels of Spaans.
Je leest de letters één voor één in en je zet de codes om in Python.
Arabisch is niet het probleem, andere mensen zijn het probleem.

Alle handige tooljes die ik had gebruikt om Hedy te bouwen waren niet voorbereid op Arabisch letters en cijfers. Ik had die codes helemaal niet in detail gelezen! Ik dacht gewoon: handig, doe mij maar een variabelenaam.

← Nederlandse naam werkt

← Arabische niet

Toen ging ik pas kijken hoe het van binnen geprogrammeerd was:

In de ingebouwde grammatica stond:

```
LETTER: UCASE_LETTER | LCASE_LETTER
```

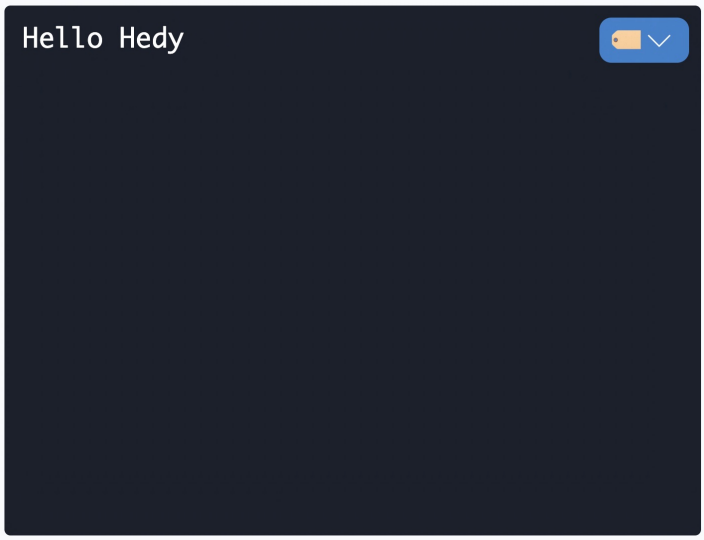
```
LCASE_LETTER: "a".."z"
UCASE_LETTER: "A".."Z"
```

Dat is dus… alleen a-z! Dan kun je dus eigenlijk niet eens alle Nederlandse woorden gebruiken. Een woord als 'kopieën' mag ook al niet, of 'répète' in het Frans.

Dit is onze nieuwe grammatica, moeilijker te lezen voor een mens, maar nu kunnen alle letters verwerkt worden!

```
LETTER: /[\p{Lu}\p{Ll}\p{Lt}\p{Lm}\p{Lo}\p{Nl}_]+/
```

Een Arabische naam werkt nu wel ↴



Nu hebben we een probleem gevonden in de linkerkant, de parser, het hakken en plakken.
Maar ook aan de Pythonkant kwamen we issues tegen.
Ik wist dit tot een jaar geleden niet, maar Arabisch heeft dus andere cijfers dan wij!

Dit noemen wij Arabische cijfers

0 1 2 3 4 5 6 7 8 9

Dit zijn Arabische cijfers

٠ ١ ٢ ٣ ٤ ٥ ٦ ٧ ٨ ٩

Het is trouwens ook best gek dat ik dat niet wist! Waarom leren we op school niet meer over andere talen en getallen?
Waarom moet volgens de Nederlandse wet ieder kind Romeinse cijfers kennen? Zodat we kunnen zien hoe oud een gebouw is? Om klok te kijken?

٠ ١ ٢ ٣ ٤ ٥ ٦ ٧ ٨ ٩

← Gebruikt door 300 miljoen mensen.

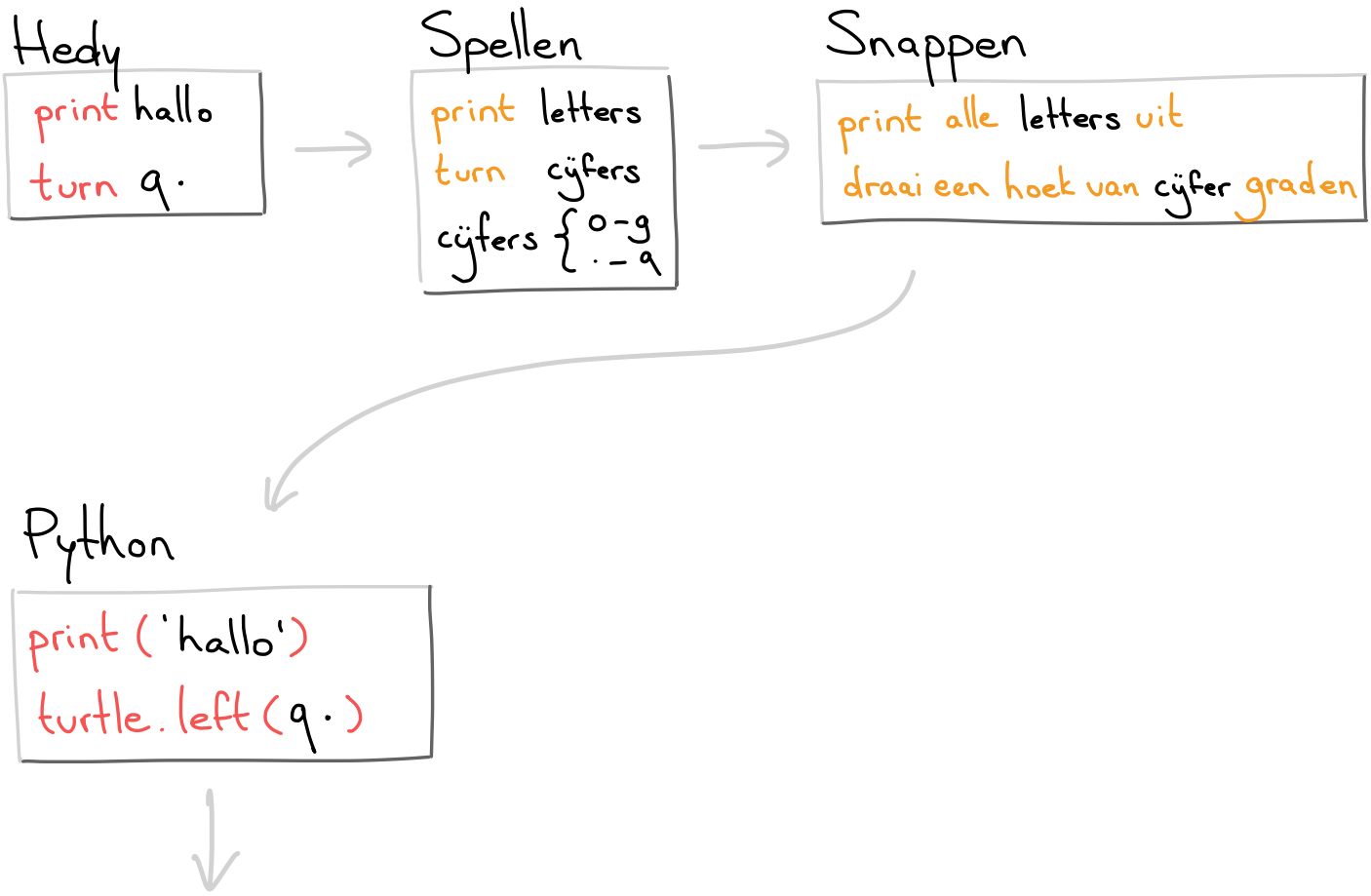
Leer je niet op school.

I II III IV V VI VII VIII IX

← Gebruikt door 0 mensen.

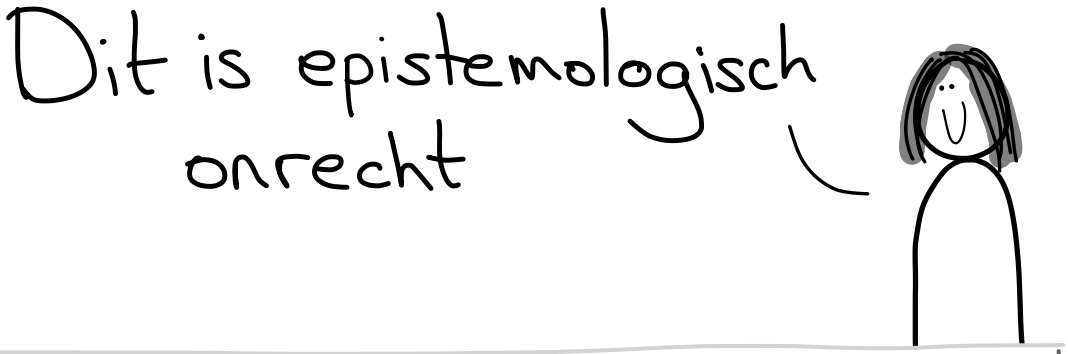
Verplicht leren!

Die cijfers inbouwen in de parser (het spellen) was niet eens zo heel moeilijk, in plaats van alleen 0 tot 9 programmerden we gewoon . tot 9 erbij.

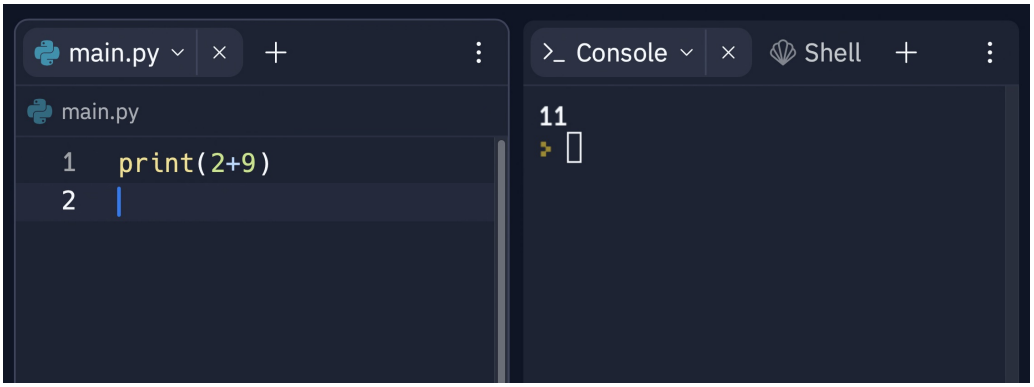


Maar dan kwam er aan de Pythonkant ook 9. uit in plaats van 90. En dat gaat dus niet; een 9 of . kan Python niet goed verwerken!

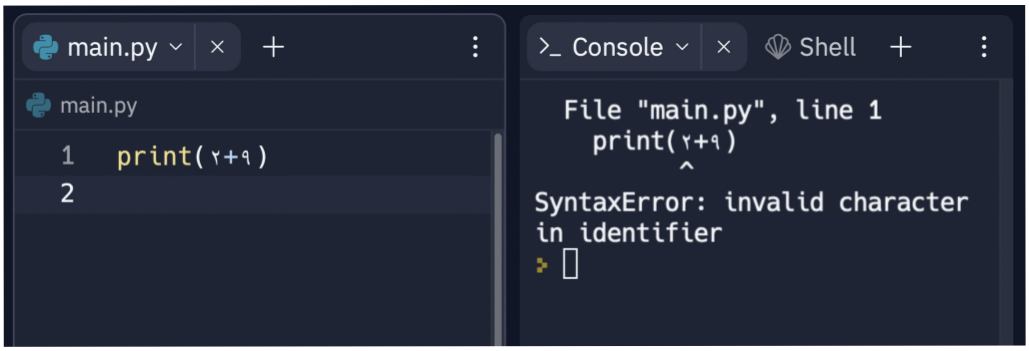
Nu kan je denken: ja dan kan iemand toch "gewoon" een 9 typen ipv een ٩? Of een ٧ in plaats van een 7? Maar als we een menselijke programmeertaal maken, dan moet dat voor alle mensen zijn! We hebben het eerder al gehad over het didactische en ergonomische aspect van programmeren in je eigen taal, maar er is ook een filosofisch argument te maken, het best beschreven door filosofe Miranda Fricker.



Epistemologie is de wetenschap van de kennis, en "epistemologisch onrecht" doet zich voor als niet alle mensen even eerlijk mogen meedoen in het maken en uitdrukken van kennis.

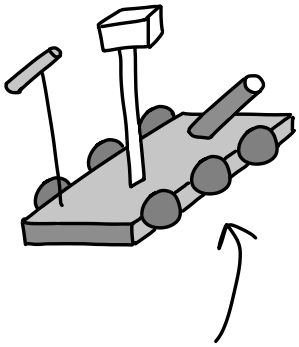


"Mijn" getallen doen het



Andermans getallen zijn "invalid characters"

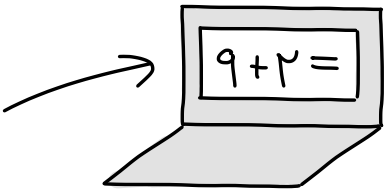
Wie is Python om te bepalen dat een getal dat voor 300 miljoen mensen hun getal is, geen getal is? De kennis en ervaring van Arabisch schrijvers (en Chinees schrijvers en Hindi schrijvers en ga zo maar door…) is niet meegenomen in het maken van programmeertalen.



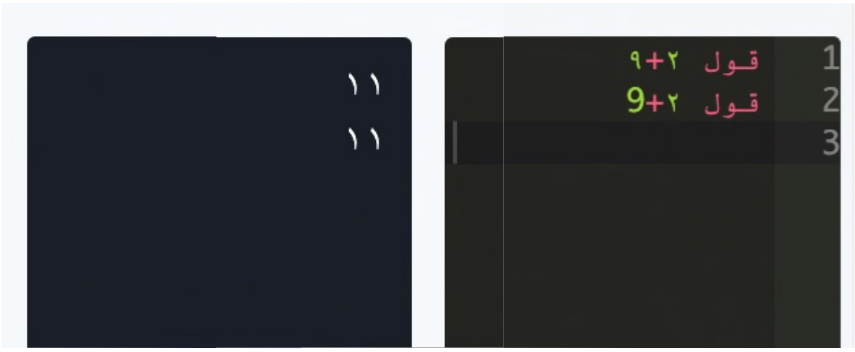
We kunnen een robotje op Mars laten rijden met software, maar we kunnen niet mogelijk maken dat iedereen zijn eigen getallen kan invoeren in de computer. Dat is soms onwil, en soms onwetendheid, maar droevig is het wel.

Doet het wel

Doet het niet



We moesten een paar trucjes uithalen om de Arabische getallen wel in Python te stoppen, maar het is gelukt! Hedy ondersteunt 30 getalsystemen en je mag ze zelfs met elkaar mixen als je wilt.

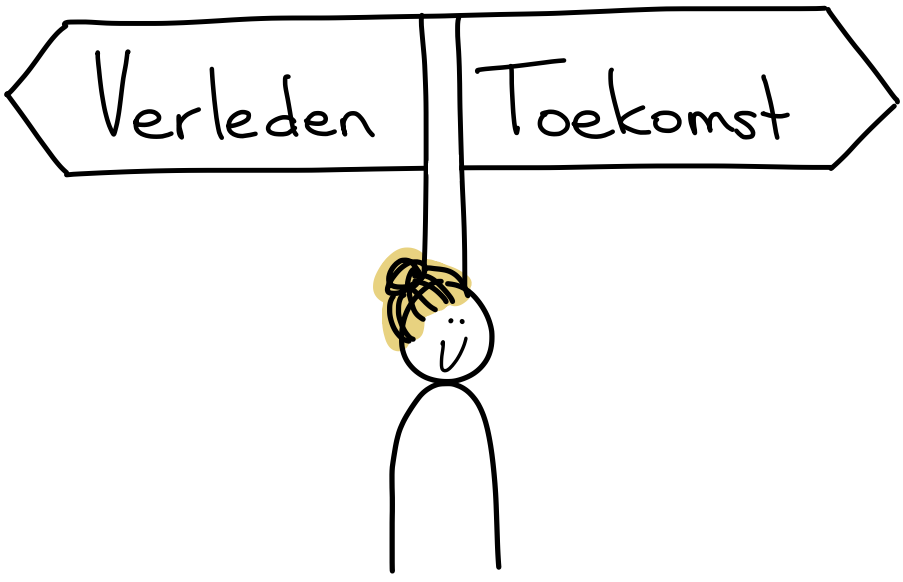


Voor zover ik weet is Hedy de enige programmeertaal die de verschillende soorten van getallen goed ondersteunt, zelfs andere programmeertalen die Arabisch ondersteunen zoals Scratch of Snap! werken niet goed met Arabische getallen.

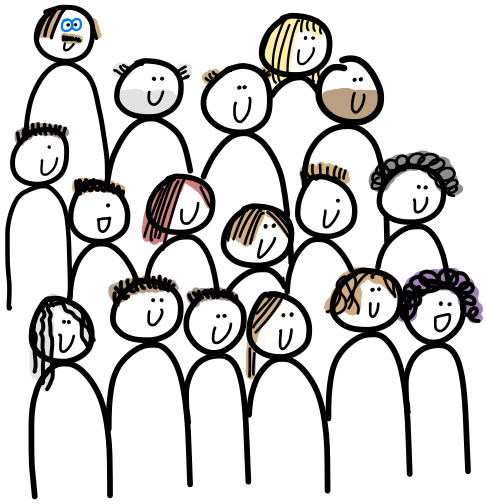
Hedy is, dankzij de vrijheid om te proberen op school, dankzij meer dan 300 contributors en vertalers, dankzij mensen zoals Alaaeddin die blijven strijden voor meer inclusiviteit voor niet Westerse talen, uitgegroeid van een klein prototype tot een wereldwijd product, met honderdduizenden gebruikers per maand.

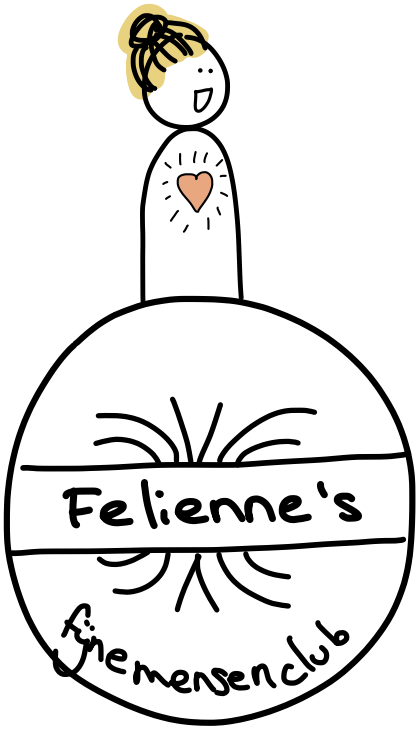
En, Hedy is zowel een product van onderzoek, als een manier voor mij om te blijven ontwikkelen en ontdekken!

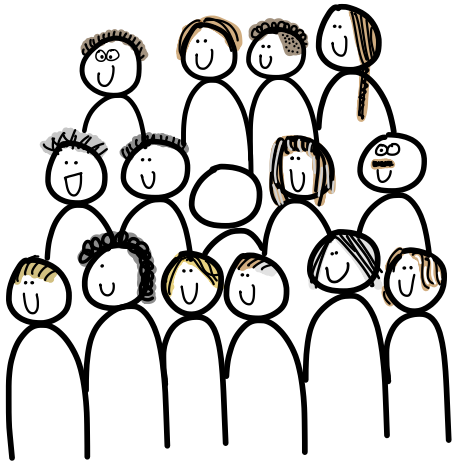
We komen aan het eind van de oratie, tijd dus om vooruit te gaan kijken. Maar eerst…



Ik had hier niet kunnen staan zonder de hulp en steun van enorm veel mensen.
Vrienden, familie, collega's, leerlingen en studenten, mentoren en docenten.





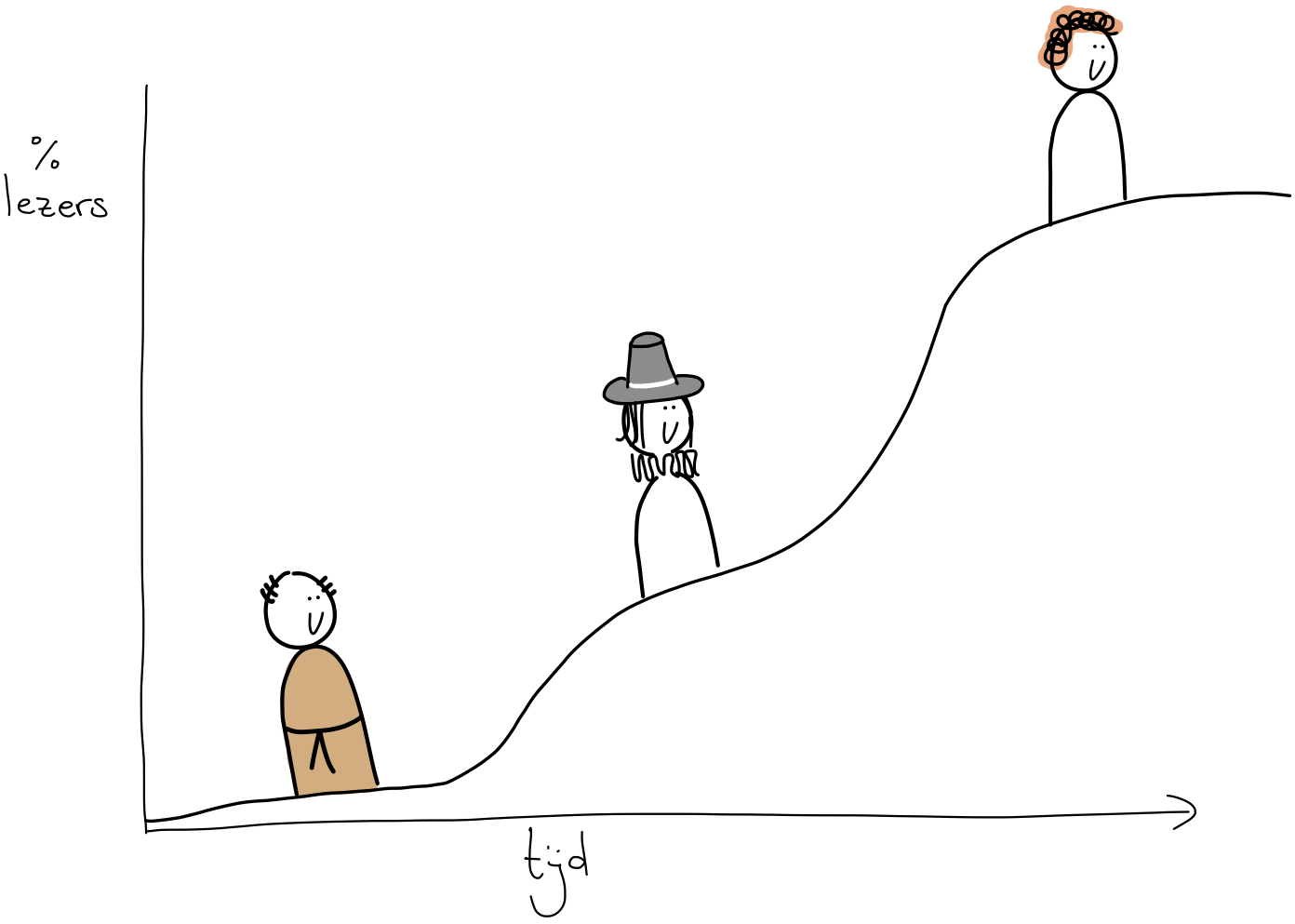


BEDANKT

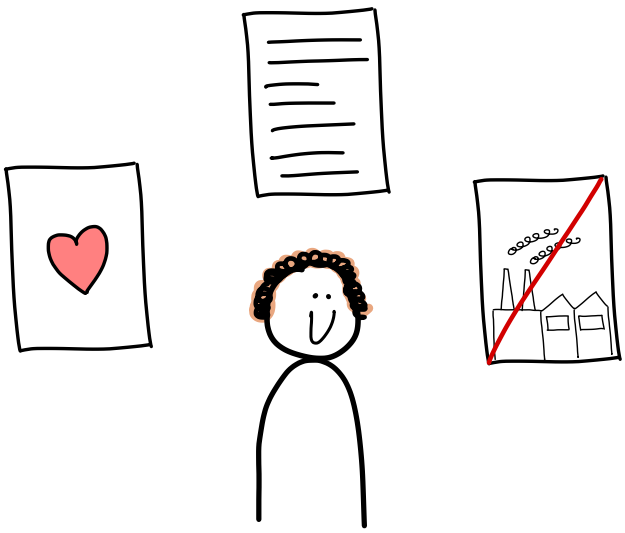
De belangrijkste staan hierboven getekend, heb je jezelf al gevonden?
Ben ik je toch vergeten..? Teken jezelf dan in het lege poppetje!

Behalve deze inner circle, moet ik moet mijn dank uitspreken voor groepen waar ik deel van heb uit mogen maken en waar ik troost en zusterschap vond! Bedankt aan alle lieverds van steungroep De waardeloze wetenschappers, handwerkcollectief Het losse steekje, Old girls network De croquettenclub, Dev- en Exnology, Joy of Coding en natuurlijk de Digibeten!

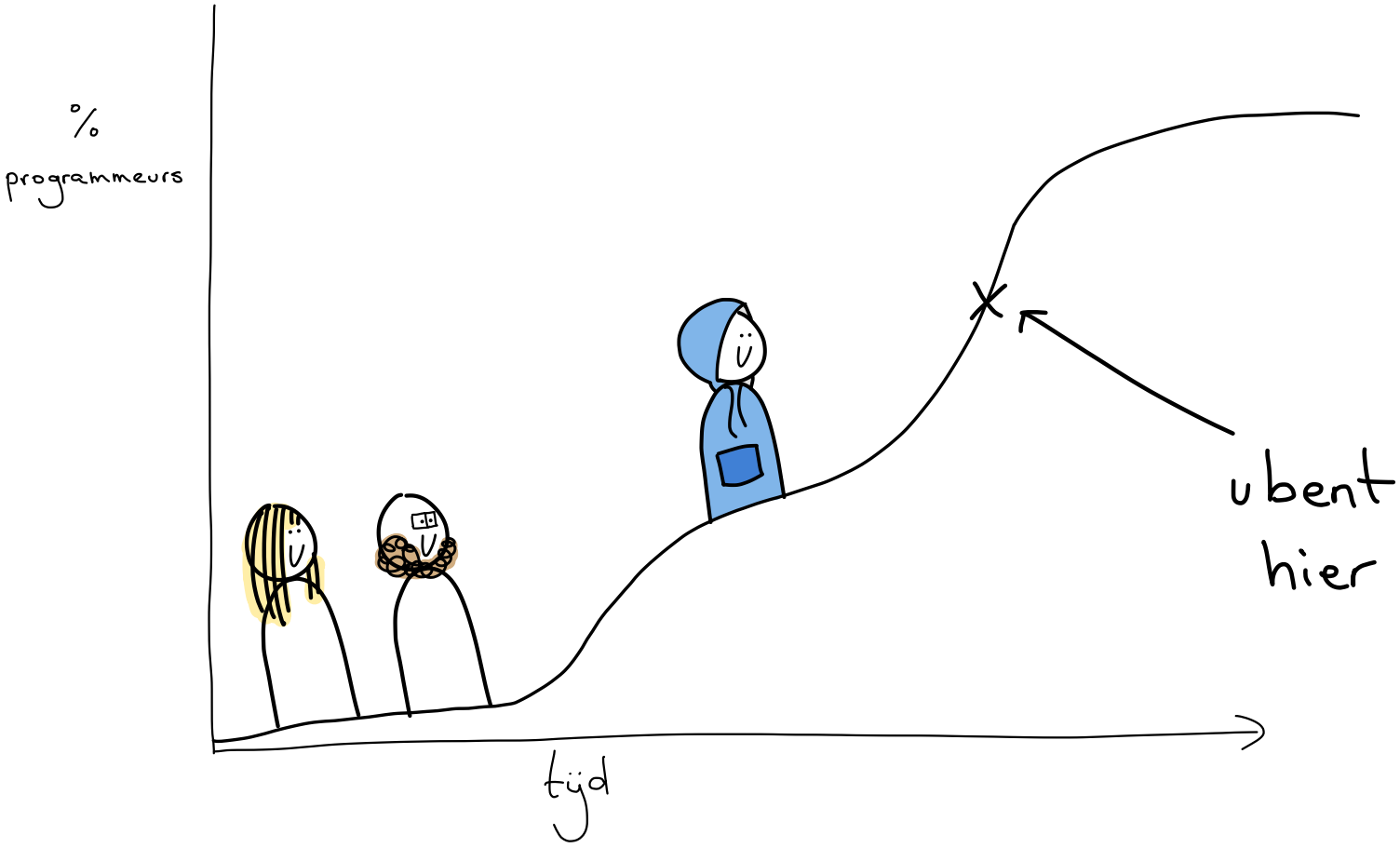
En dan, de toekomst! Lang geleden, kon er bijna niemand lezen en schrijven, in onze cultuur vooral monniken. Dat zag je aan de dingen waarover geschreven werd.
Later konden meer mensen schrijven, maar vooral de elite, en nog later bijna iedereen.



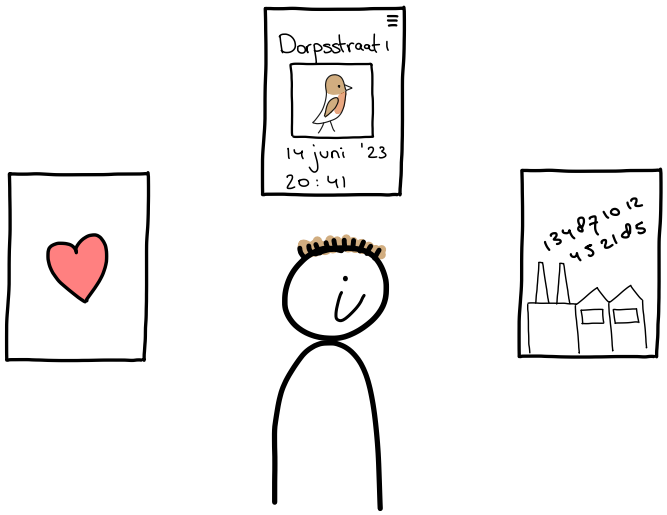
En natuurlijk hoeft niet iedereen Agatha Christie te worden, maar iedereen kan wel lezen en schrijven wat hij of zij maar wil: een liefdesbrief, een brief naar de gemeente of een pamflet tegen vervuilende industrie.



In programmeren zijn we nu misschien net de monnikentijd voorbij: programmeren heeft de stap al gemaakt van voornamelijk vrouwen, toen programmeren als secretaressewerk gezien werd, naar heel klein beperkt groepje wetenschappers, tot een iets grotere groep programmeurs, die niet altijd maar wel vaak uit elite bestaat. We gaan nu, door allerhande veranderingen naar een wereld waarin iedereen een beetje kan programmeren.

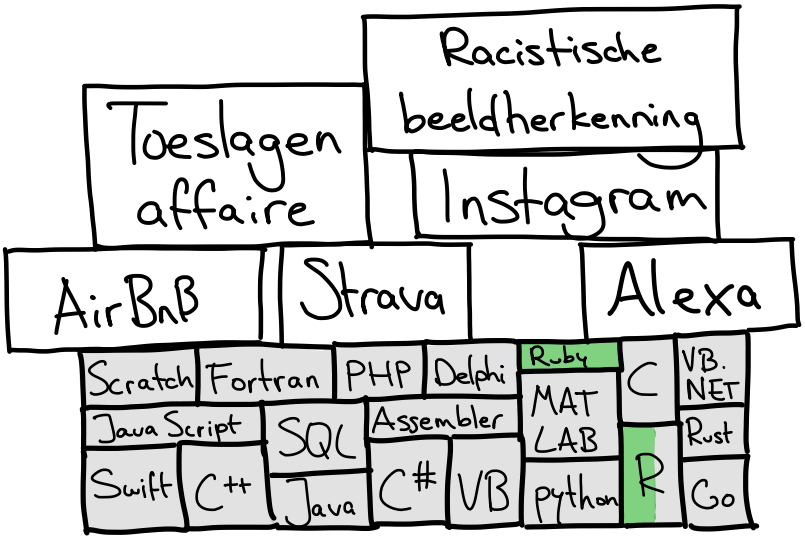


En net als bij schrijven gaat het niet over professionals opleiden, maar over laten zien dat je programmeren kan gebruiken voor allerhande taken.
Een app voor een geliefde, een roodborstjes-telapp, of een data-analyse van vervuilende industrie.



Ik kijk uit naar alle geweldige software die zo veel mensen in de toekomst gaan maken.
Maar, ik maak me ook zorgen, want heel veel software is niet inclusief. We kennen allemaal de toeslagenaffaire of beeldherkenning die zwarte mensen minder goed ziet.
Of de bekendste sportapp Strava die wel 33 soorten fietsonderdelen kan tracken, maar geen beha's, iets dat de helft van de mensen op de wereld gebruikt! En waarom zijn assistenten zoals Alexa altijd vrouwelijk, maar herkennen ze vrouwenstemmen juist minder goed?

En Instagram, flitsbezorgers en AirBnB… Wie wordt daar rijk van en wie armer?
Wiens wijk verandert erdoor?



Er zijn heel veel redenen voor die problemen, heel veel ervan buiten software.
Maar binnen software kunnen we zien dat bijna alle apps en programma's leunen op een klein setje programmeertalen, bijna exclusief gemaakt door Westerse mannen.
Van de meestgebruikte 20 talen is er maar één gemaakt door een Japanse man, en eentje mede door een Pacific Islander.

De andere 18 meestgebruikte programmeertalen zijn gemaakt door witte mannen, die maar 8 procent van de mensheid zijn! De andere 92% van de mensen heeft dus niet eens mogen meedenken over het maken van de talen aan de basis van zo goed als alle software.
Dan is het niet gek als de inclusiviteit van die software ook te wensen overlaat.

Dat wil niet zeggen dat alle bestaande talen slecht zijn natuurlijk! Sommige van mijn favoriete talen zijn gemaakt door een man, nog wel een Nederlandse.

Maar het betekent wel dat de ervaring van veel mensen niet is meegenomen.
Als je cijfers niet 0 tot 9 zijn, dan is er niet over jou gedacht.
Als je programmeren spannend vindt, dan is er niet over jou nagedacht.

Kijk bijvoorbeeld eens naar dit "simpele voorbeeldje" in C++

```cpp
#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

Zou je code zo maken als je denkt aan meiden die vóór hun eerste les al 10 jaar horen dat programmeren voor hen te moeilijk is? Zou je code zo maken als je denkt aan iemand die het met een schermlezer leest, en dus *'std dubbele punt dubbele punt cout kleiner dan kleiner dan'* hoort? Waarschijnlijk niet!

Heel veel doelgroepen zijn niet meegenomen en daarom zijn programmeertalen voor hen vaak ontoegankelijker.

Dus als mensen mij vragen wat ik komende tijd ga doen, dan is mijn antwoord dat ik me hard ga maken voor programmeertalen die vanaf de basis voor iedereen zijn, met Hedy als voorbeeld en als speeltuin.