Van code begrijpen naar code schrijven

06-Nov-2025

Renske Weeda

Radboud Universiteit

WIE HERKENT DIT?

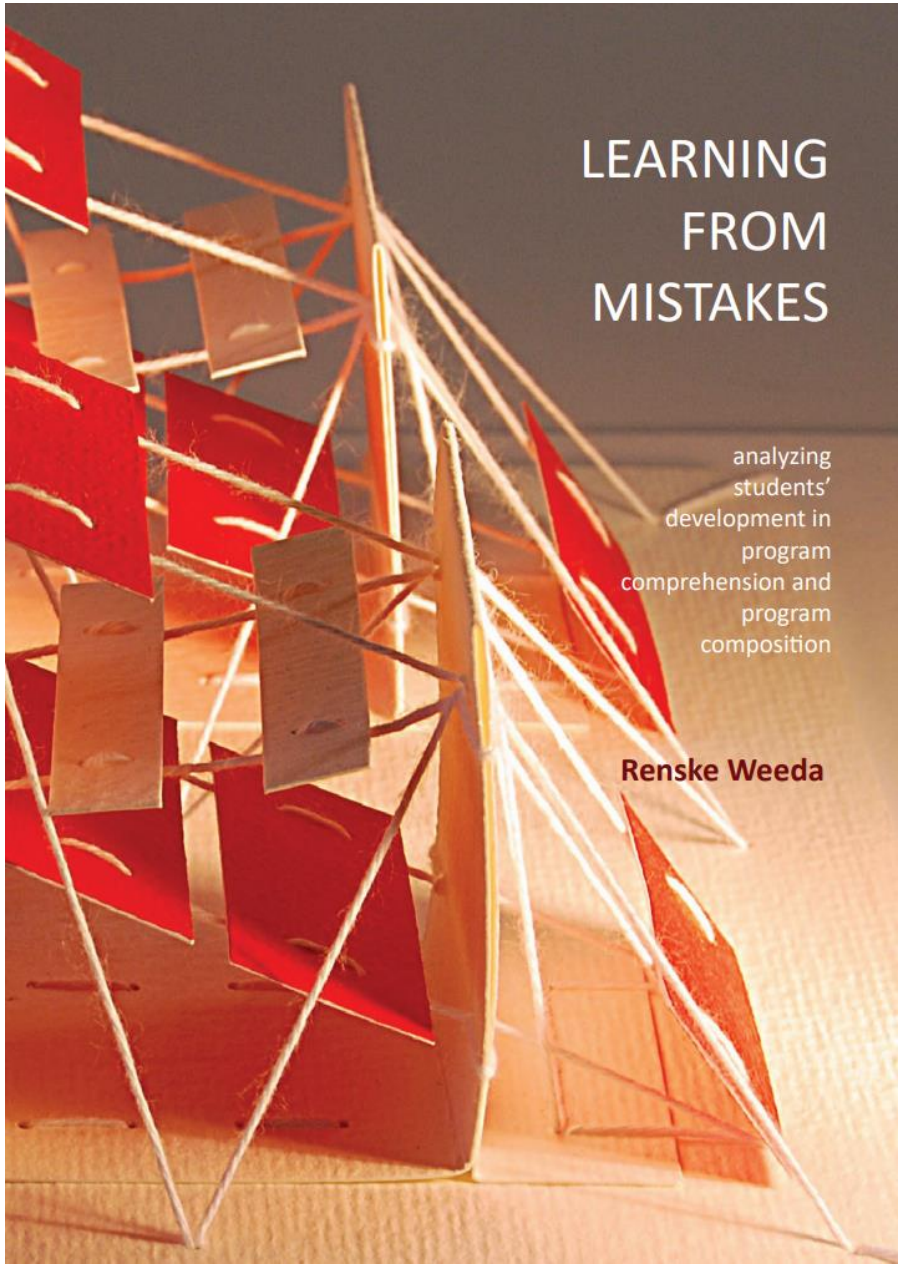
- Leerlingen doorlopen de lesstof
- Je hebt het idee dat ze de stof kennen
- Dan bij een PO blijkt dat ze veel hulp nodig hebben van ChatGPT?
- Teken dat ze onzeker zijn en wij het als docenten niet goed doen!

- Hoe kunnen we strategieen aanpassen zodat leerlingen het leuk vinden om te leren en blijven leren?

- Sterke fundament basiskennis is belangrijk om op verder te bouwen en **zelfvertouwen**.
- Uitbouwen in kleine stappen: het begint bij begrip van code, voordat je het je eigen kan maken!
  het vermogen om code te schrijven is gerelateerd aan mate van code begrip

VOORSTELLEN

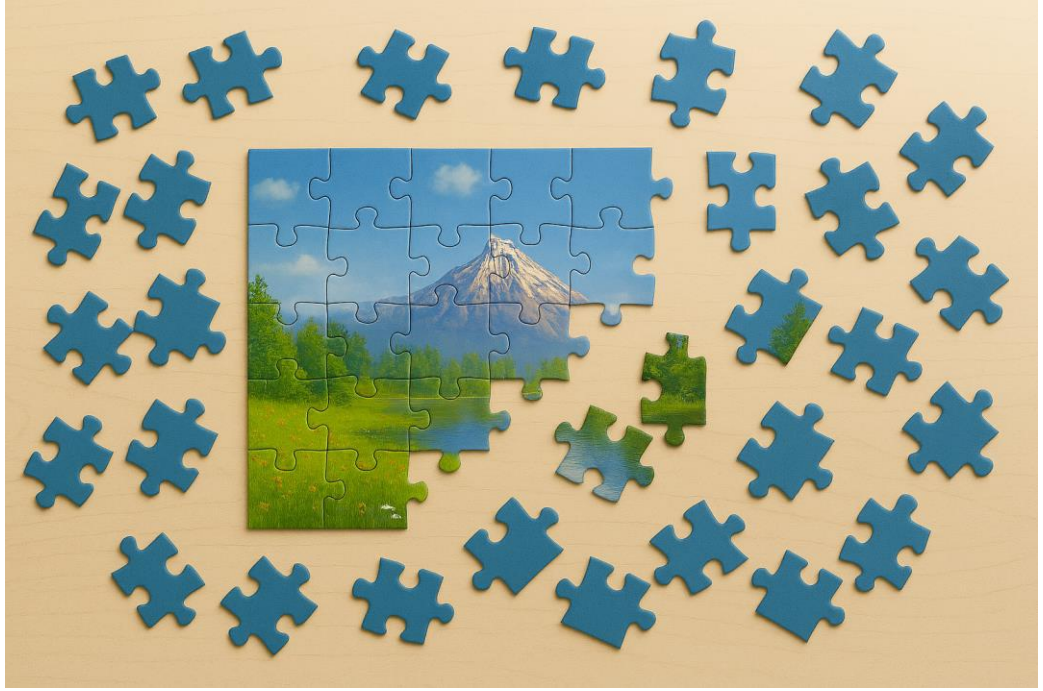
- Renske Weeda
  - Gepromoveerd programmeeronderwijs VO
    - De ‘waarom’ achter fouten & moeilijkheden bij programmeren
  - Docent informatica Montessoricollege Nijmegen
  - Auteur en bestuurslid Informatica-Actief
  - Vakdidacticus Co-Teach



INHOUD

- Code begrip: wat en waarom
- Plannen: vaak-gebruikte aanpakken
- Variabelrollen: 9 veel-voorkomende rollen van variabelen
- Semantic waves
- Begrip: blok model
- Voorbeeld opdrachten uit blok model
- PRIMM: van begrip naar schrijven

LEREN PROGRAMMEREN BEGINT BIJ CODE BEGRIJPEN



- lezen ≠ begrijpen
- Begrip = abstracte manier redeneren
- Begrip over een blok code -> ‘chucking’
  - sterke mentaal model
  - abstracte manier redeneren
  - structuren herkennen (zoals plannen en patronnen)
  - verlaagt cognitieve belasting (bos zien, niet alleen de bomen)

- Essentieel om kennis **flexibel** toe te passen

PLANNEN: VEEL GEBRUIKTE AANPAKKEN

SOLOWAY (1986), DE RAADT (2009)

 Vraag-verwerk-print	 Zoeken	 Max/Min	 Som/Gemiddelde	 Sorteren	 Filter
Lees gegevens in, verwerk, en toon het resultaat	Komt een element voor?	Zoek het grootste / kleinste	Tel aantal, of bereken som / gemiddelde	Rankgschikken	Filter (on)gewenste gegevens

WAAROM PLANNEN - ‘VEEL GEBRUIKTE AANPAKKEN’?

- Een mentale **bibliotheek** van **aanpakken**: toepassen en aanpassen
- Herkennen van plannen (zoals *zoeken*, *filter*, *som*) helpt leerlingen om **te starten** bij een opdracht
- Minder **misconcepten** en versnellen het schrijven van code.
- Bevorderen **algoritmisch denken**: complexe problemen opdelen in beheersbare stappen
- Ondersteunen ‘**chunking’** → minder geheugenbelasting, meer ruimte voor redeneren.
- Leerlijn: plannen bouwen voort op elkaar (*tellen* → *filter* → *som* → *gemiddelde*).

De 9 meest gebruikte variabelrollen

(Sajaniemi, 2005)

Vaste waarde

Een constante

```
PI = 3.14
```

Stapper

Neemt systematisch opeenvolgende waarden aan (bijv. teller in een lus).

```
i++
```

Wandelaar

Loopt door een datastructuur (lijst/array) heen.

```
for element in lijst:
```

Meest recente houder

Slaat de meest recent verwerkte waarde op.

```
recente_invoer = input()
```

Meest gewenste houder

Bewart de 'beste' waarde tot nu toe (bijv. max/min).

```
beste = huidig_element
```

Verzamelaar

Verzamelt of telt waarden op (bijv. som/aantal).

```
totaal += huidig
```

Container

Groepeert meerdere waarden

```
resultaten = [2,5,1]
```

Tijdelijke variabele

Tijdelijk voor tussenresultaten (bijv. wisselen/berekening).

```
temp=a; a=b; b=temp
```

Boolean vlag

Booleaan die éénmaal van toestand verandert om een gebeurtenis aan te geven.

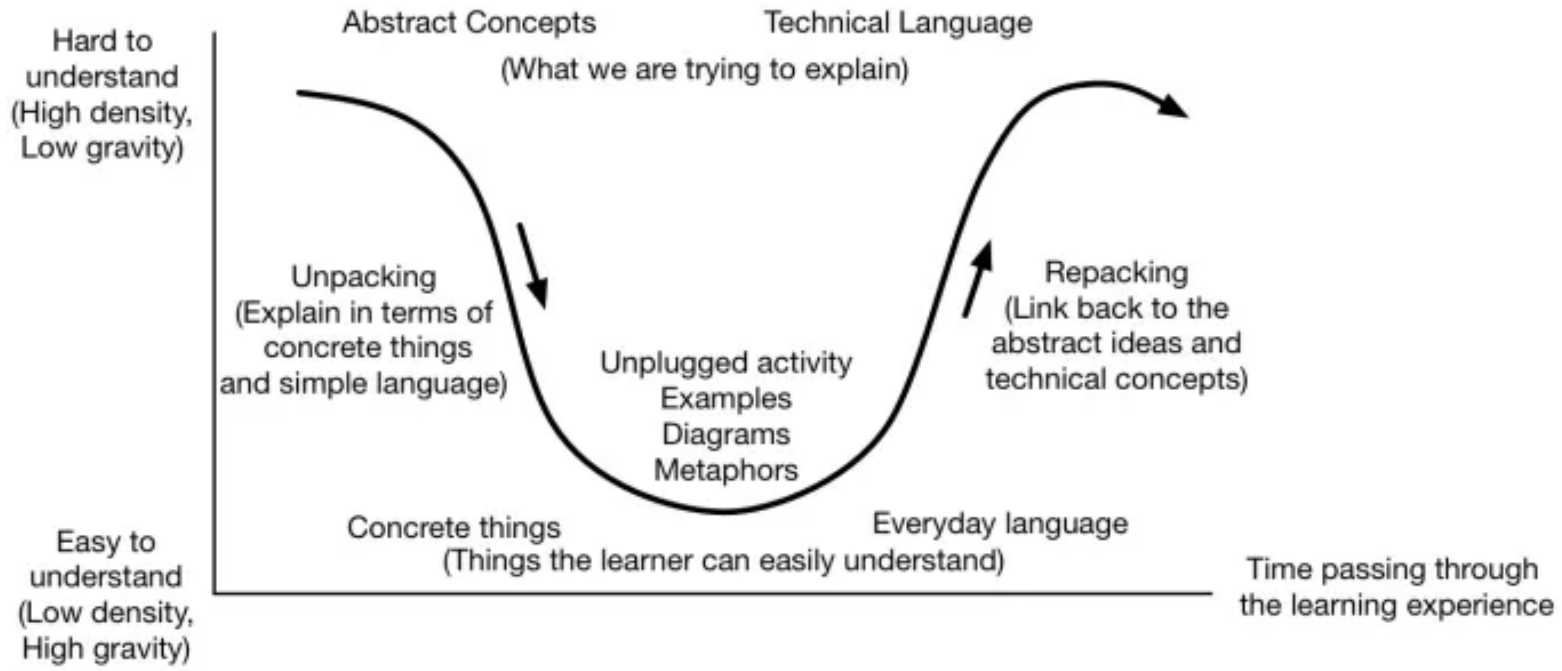
```
gevonden = True
```

WAAROM VARIABELROLLEN?

- Ondersteunen het **herkennen van plannen** en gedragspatronen in code.
- Helpen bij **abstract redeneren**: het **doel** van een variabele begrijpen ipv. de code regel-voor-regel te lezen.
- Bevorderen '**chunking**' – code wordt opgedeeld in betekenisvolle gehelen, wat cognitieve belasting verlaagt (Ben-Ari & Sajaniemi, 2004).
- **Gemeenschappelijke taal** om code te bespreken (bijv. stapper, verzamelaar).
- Brug tussen lezen en schrijven van code — essentieel voor programmeervaardigheid.

SEMANTIC WAVES

(KARL MATON, 2013)



- Leren wordt sterker door wisselen tussen concreet (voorbeelden) en abstract (begrippen / principes)
- Leid tot dieper begrip en transfer

OPDRACHTVARIATIES OP BEGRIP A.D.H.V. BLOK MODEL (SCHULTE, 2008)

Verschillende aspecten van code begrip:

- **omvang**: klein / gedetailleerd vs. groot / overzicht (*rijen*)
- cognitieve activiteit: **structuur** – **uitvoer** – **betekenis** (*kolommen*)

Macro	Algemene structuur	Programma uitvoer	Doel van programma
Relationeel	Relaties tussen methoden	Volgorde van uitvoering code	Bijdrage van subdoelen aan hoofdoelen
Blok	Bij elkaar horende regels	Uitvoer van een codeblok of methode	Doel van een codeblok
Atomisch	Basis elementen	Uitvoer van coderegel	Doel van coderegel
	Structuur	**Uitvoer**	**Betekenis**

BLOK MODEL – OPDRACHTEN PER CELL

Macro	Bepaal hoe blokken genest zijn	Haal overbodige code weg (bv. blok dat nooit uitgevoerd kan worden door if-voorwaarde)	Doel van programma samenvatten
Relationeel	Scope van een variabele aangeven	Relateer code en stroomdiagram (vul aan)	Plan herkennen
Blok	Bepaal welke regels bij een blok/statement horen (bv else)	Parson's: rangschikken gehusselde code van een blok	Doel van een blok code aangeven
Atomisch	Identificeer statements/variabelen	Waarden traceren	Rol variabele benoemen
	Structuur	**Uitvoer**	**Betekenis**

WAAROM SAMENVATTEN?

- Samenvatten: “Doel van een stuk code”
- Sterke correlatie tussen code lezen en code schrijven *(Lister,2004)*
- Met een samenvatting tonen leerlingen aan:
  - begrip
  - misconcepties
  - abstractieniveau
- Korte/snelle opdrachtjes
- Ideaal voor gerichte feedback

Macro	Algemene structuur	Programma uitvoer	Doel van programma
Relationeel	Relaties tussen methoden	Volgorde van uitvoering code	Bijdrage van subdoelen aan hoofdoelen
Block	Bij elkaar horende regels	Uitvoer van een codeblok of methode	Doel van een codeblok
Atomisch	Basis elementen	Uitvoer van coderegel	Doel van coderegel
	Structuur	**Uitvoer**	**Betekenis**

BLOK-MODEL VOORBEELDOPDRACHTEN

Structuur:

1) Hoeveel variabelen tel je?

2) Wat is de scope van `b`?

Uitvoer:

3) Hoe vaak `while`-blok uitgevoerd?

4) Wat gebeurt er in de regel `a+=b`?

Betekenis:

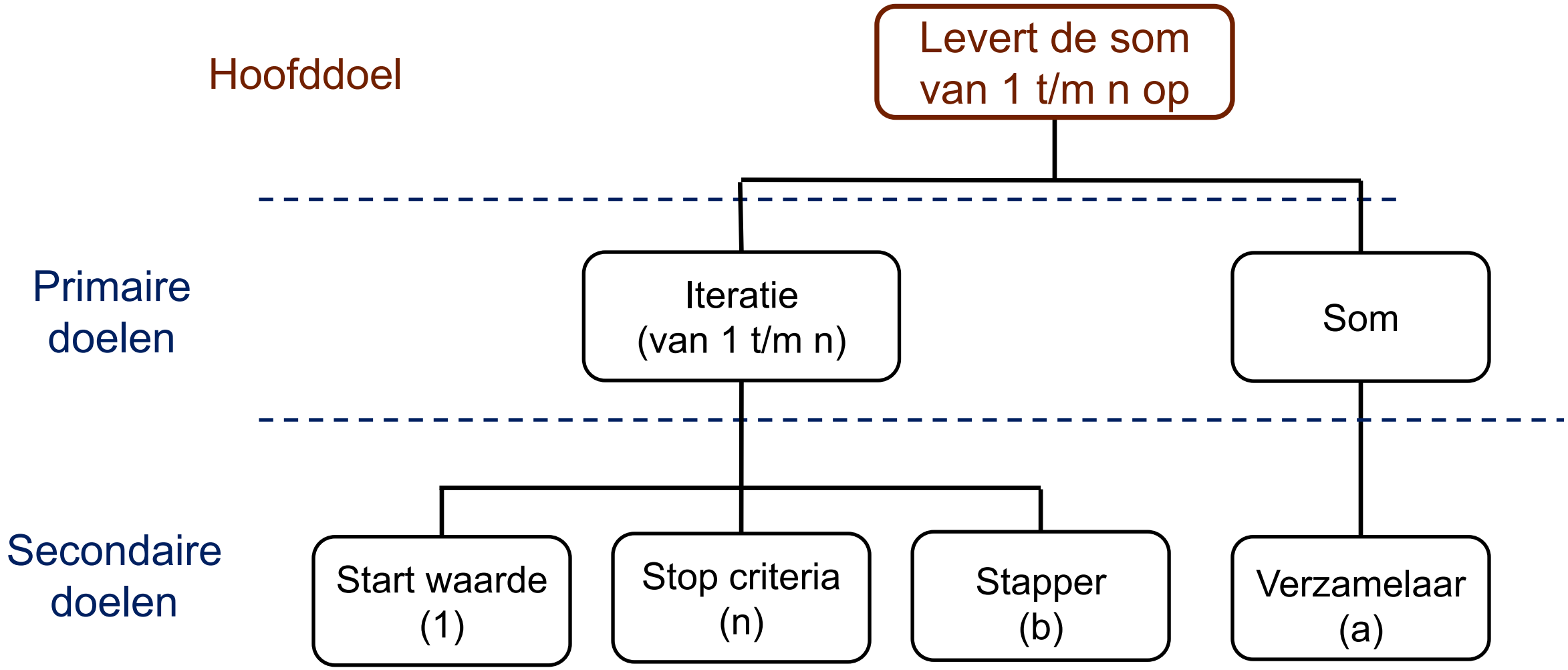
5) Welke variabel rollen herken je?

6) Welke plan herken je?

7) Wat is het doel van het programma?

```
def doeIets(n):
    a = 0
    b = 1
    while b <= n:
        a += b
        b = b + 1
    return a
```

1) 3 vars: a,b,n
2) zie rechthoek
3) n keer
4) **a** (som) wordt opgehoogd met **b**
5) n: vaste waarde
   b: stapper (doorlopt waarden)
   a: verzamelaar (som)
6) som-plan
7) levert de som van 1 t/m n op



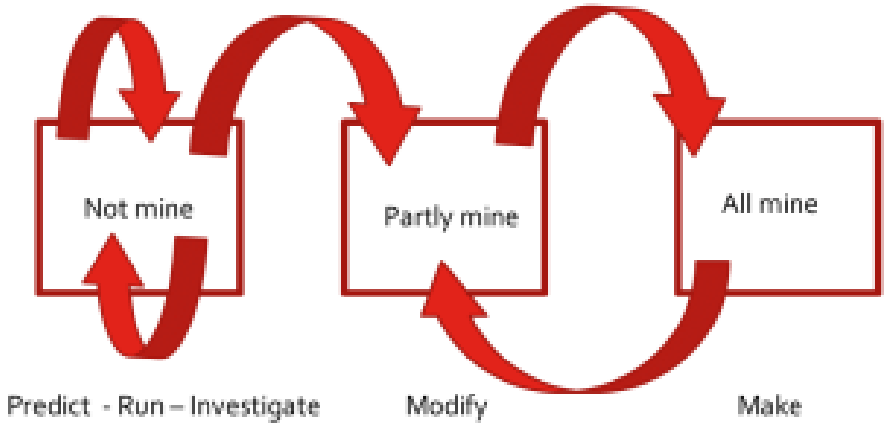
Weeda et al., 2020 (Koli Calling)

WAT IS DOEL VAN DEZE CODE?

```
def doeIets(n):
  a = 0
  b = 1
  while b <= n:
    a += b
    b = b + 1
  return a
```

Niveau	Voorbeeld van antwoord	Toelichting
Abstract en correct	*"Het print de som van 1 t/m n."*	Goede samenvatting
Abstract, niet correct	*"Telt getallen 1 tot n op."*	Mist 't/m n'
Abstract, niet compleet	*"Print getallen van 1 t/m n."*	Mist opsomming, print vs. opleveren
Niet abstract, niet compleet	*"Telt steeds b bij a op, en dan b plus 1."*	Regel-voor-regel 'voorgelezen', toont geen begrip
Fout / misconceptie	*"Als b kleiner gelijk aan n, dan "*	Misvatting over while loop
Gegokt	*"Het doet iets"*	Gegokt: Abstract en correct ☺, maar niet compleet

PRIMM ONDERZOEKEND LEREN



Predict: Wat denk je dat deze code doet?

Run: Run het en reflecteer op je voorspelling

Investigate: Traceer, annoteer, label variabellen, debug

Modify: Pas aan, breid uit

Make own: Maak eigen nieuwe programma (evt. met hergebruik)

```
lijst = [25,70,70,70,0,34,76]

a = 1
b = lijst[0]

for i in lijst:
    if lijst[i] > b:
        b = lijst[i]
        a = 1
    elif lijst[i] == b:
        a += 1

print(a)
```

PRIMM: ONDERZOEKEND LEREN

Predict	
Run	
Investigate	Haal de fout eruit. Hernoem variabelen. Maak testcases.
Modify	Maak er een functie van. Verzamel posities van max. Vereenvoudig (zonder indices)
Make	Playlist: Top 3 populairste nummers Fitbit: Hoe vaak record stappen gehaald

```
lijst = [25,70,70,70,0,34,76]

a = 1
b = lijst[0]

for i in range(len(lijst)):
    if lijst[i] > b:
        b = lijst[i]
        a = 1
    elif lijst[i] == b:
        a += 1

print(a)
```

ZELF OPDRACHTVARIATIES MAKEN M.B.V. BLOK MODEL

- Omvang: regel vs blok vs programma
- Structuur / Uitvoer / Betekenis

```
lijst = [25,70,70,70,0,34,76]

a = 1
b = lijst[0]

for i in range(len(lijst)):
    if lijst[i] > b:
        b = lijst[i]
        a = 1
    elif lijst[i] == b:
        a += 1

print(a)
```

OPDRACHTVARIATIES M.B.V. BLOK-MODEL

Structuur

- Hoeveel variabelen tel je?
- Geef de scope van `if` aan.
- Geef de scope van b aan.

Uitvoer:

- Wat gebeurt er bij `lijst[i]>b`?
- Wat gebeurt er in de `if`-blok?

Betekenis:

- Welke variabel rollen herken je?
- Welke plannen herken je?
- Wat is het doel van het programma?

```
lijst = [25,70,70,70,0,34,76]

a = 1
b = lijst[0]

for i in range(len(lijst)):
    if lijst[i] > b:
        b = lijst[i]
        a = 1
    elif lijst[i] == b:
        a += 1

print(a)
```

SAMENVATTEND

- **Beter begrip → beter schrijven**
- Herkennen helpt ‘starten’ en aanpassen:
  - 9 **variable rollen**
  - **Plannen**: veel gebruikte aanpakken
- blok-model m.b.t. code begrip
  - structuur – uitvoer – betekenis
  - klein onderdeel → blok/programma als geheel
- Code samenvatten: korte opdrachtjes – veel inzicht
- Van lezen naar schrijven: **P**redict – **R**un – **I**nvestigate – **M**odify – **M**ake own

Vragen / opmerkingen:
renske.weeda@ru.nl

BRONNEN

- Sajaniemi, J. (2002). Roles of variables in novice-level procedural programs.
- Sajaniemi, J., & Navarro-Prieto, R. (2005). Roles of variables in experts' programming knowledge.
- Ben-Ari, M., & Sajaniemi, J. (2004). Metaphors and beacons for variable roles.
- PRIMM: https://primmportal.com/
- Sentance, S & Waite, J 2017, PRIMM: Exploring pedagogical approaches for teaching text-based programming in school. in *Proceedings of the 12th Workshop in Primary and Secondary Computing Education: WIPSCE '17.* Nijmegen. DOI: 10.475/123_4
- Schulte, C., 2008. blok model: An educational model of program comprehension as a tool for a scholarly approach to teaching, in: Proceedings of the Fourth International Workshop on Computing Education Research, New York, NY, USA: ACM, ICER '08
- Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Mostrom, J.E., Sanders, K., Seppala, O., et al., 2004. A multi-national study of reading and tracing skills in novice programmers, in: ACM SIGCSE Bulletin, ACM, vol. 36.
- Armoni, M., 2013. On teaching abstraction in computer science to novices., Journal of Computers in Mathematics and Science Teaching, 32 (3), 265-284
- Soloway, E. (1986). *Learning to program = Learning to construct mechanisms and explanations. Communications of the ACM, 29*(9), 850–858
- Denny, P., Luxton-Reilly, A., & Simon, B. (2008). *Evaluating a new exam question: Parsons problems.* In *Proceedings of the Fourth International Workshop on Computing Education Research* (pp. 113–124).
- Izu, C., Philpott, A., & Duran, R. (2019). *Learning trajectories in introductory programming: From program comprehension to composition.* In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19).*
- De Raadt, M., Watson, R., & Toleman, M. (2009). *Teaching and assessing programming strategies explicitly.* In *Proceedings of the Eleventh Australasian Conference on Computing Education – Volume 95* (pp. 45–54).