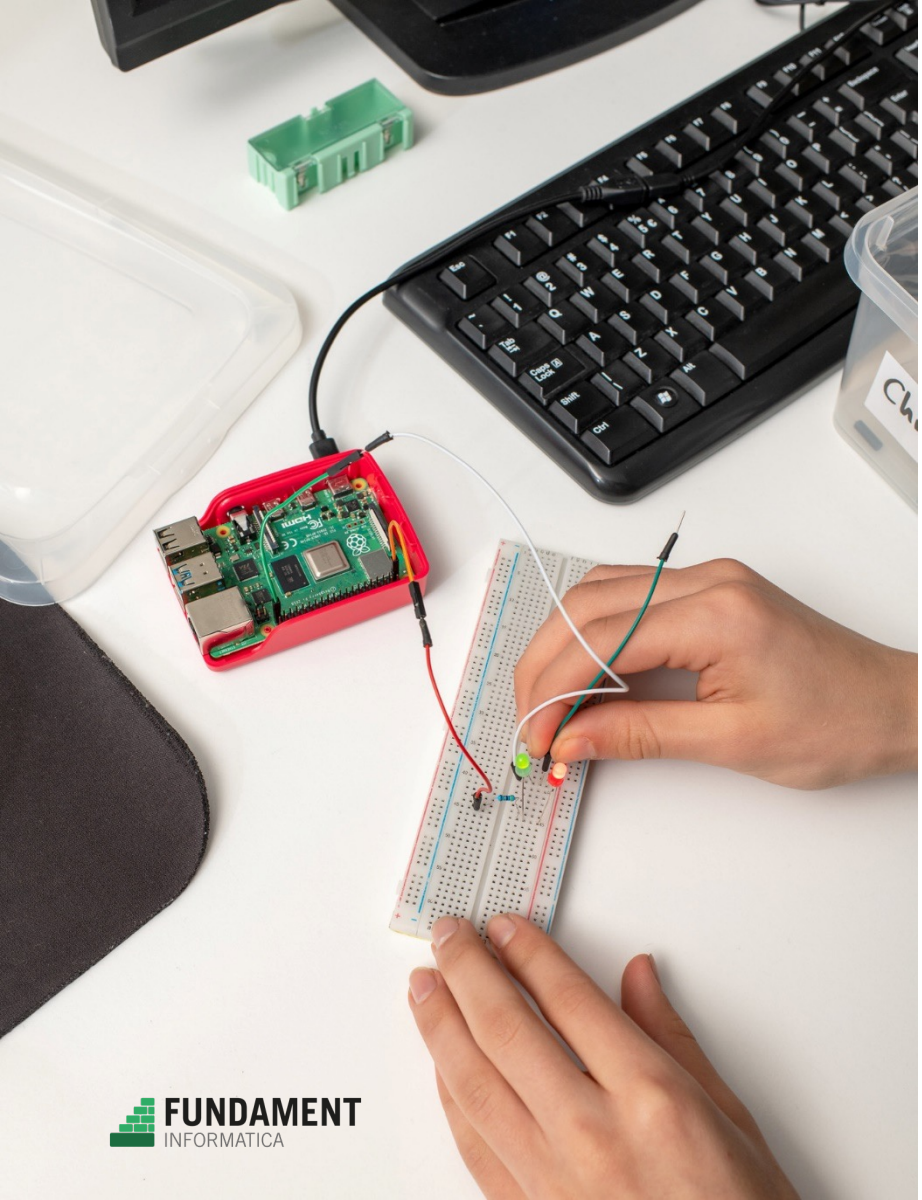


Prolog in de klas: Puzzelen met Prolog

LOGISCH PROGRAMMEREN

Sudoku

Puzzelen met Prolog

FUNDAMENT
INFORMATICA

Sudoku (Japanse puzzel)

- Logische puzzel
- Tabel van 9 bij 9 vakjes
- Gegroepeerd in 9 blokken
- Elk vakje een cijfer, met regels:
  - Alleen 1 t/m 9 toegestaan
  - Elke rij, kolom en blok:
    elk cijfer maar één keer

5		1	6		7	9		
		9			3	2	5	
8	2	7		9				
9		2		5	1	3	7	
3			9	8				
		5	7		6			
4		6		7	5		3	2
	1					7		5
		3				1	9	6

Sudoku (Japanse puzzel)

- Logische puzzel
- Tabel van 9 bij 9 vakjes
- Gegroepeerd in 9 blokken
- Elk vakje een cijfer, met regels:
  - Alleen 1 t/m 9 toegestaan
  - Elke rij, kolom en blok:
    elk cijfer maar één keer

- **Hoe los jij een sudoku op?**

5	3	1	6	2	7	9	8	4
6	4	9	8	1	3	2	5	7
8	2	7	5	9	4	6	1	3
9	6	2	4	5	1	3	7	8
3	7	4	9	8	2	5	6	1
1	8	5	7	3	6	4	2	9
4	9	6	1	7	5	8	3	2
2	1	8	3	6	9	7	4	5
7	5	3	2	4	8	1	9	6

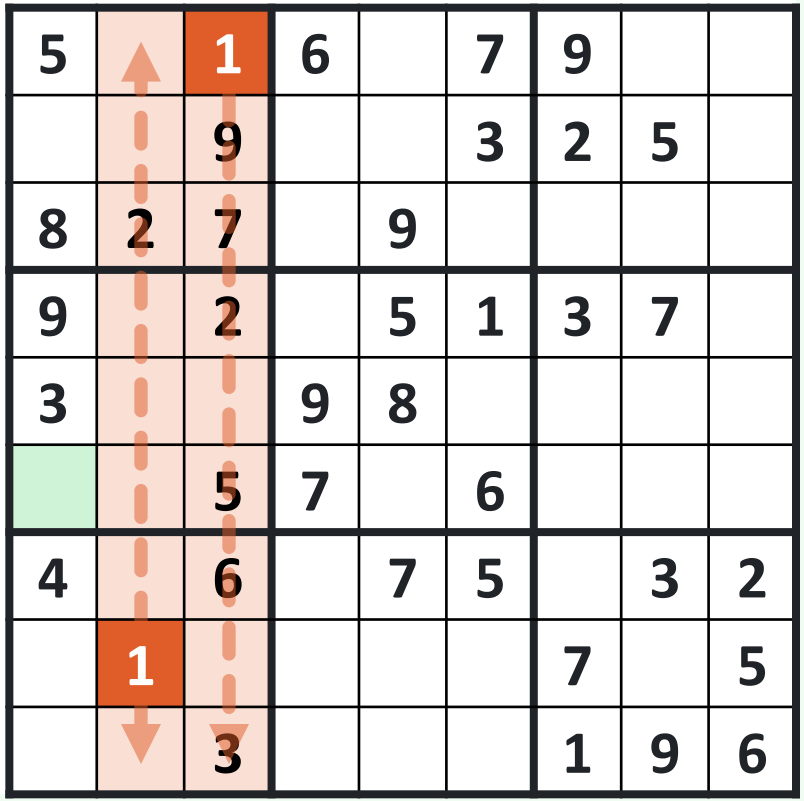
Sudoku (Japanse puzzel)

- Hoe los jij hem op?
- Welke methoden zijn er?

5		1	6		7	9		
		9			3	2	5	
8	2	7		9				
9		2		5	1	3	7	
3			9	8				
		5	7		6			
4		6		7	5		3	2
	1					7		5
		3				1	9	6

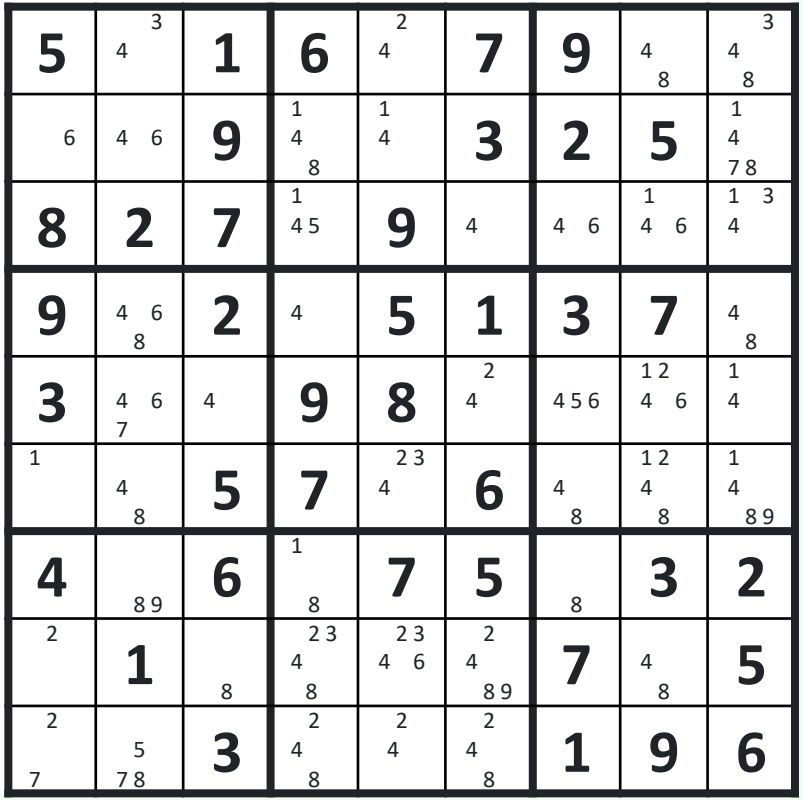
Sudoku’s oplossen

- Werk met eenvoudige tactieken:
  - Voor de hand liggende oplossingen
  - Geblokkeerde getallen in een blok
  - Hints opschrijven, tegensprekende hints wegstrepen
  - …en meer
- Hiernaast: geblokkeerde getallen



Sudoku’s oplossen

- Werk met eenvoudige tactieken:
  - Voor de hand liggende oplossingen
  - Geblokkeerde getallen in een blok
  - Hints opschrijven, tegensprekende hints wegstrepen
  - …en meer
- Hiernaast: hints opschrijven
- In totaal ± 38 tactieken



FUNDAMENT INFORMATICA

Sudoku-oplosser in Python

- Stap 1: Puzzel in Python-code

5		1	6		7	9		
		9			3	2	5	
8	2	7		9				
9		2		5	1	3	7	
3			9	8				
		5	7		6			
4		6		7	5		3	2
	1					7		5
		3				1	9	6

```
puzzel = [
    [5, 0, 1,  6, 0, 7,  9, 0, 0],
    [0, 0, 9,  0, 0, 3,  2, 5, 0],
    [8, 2, 7,  0, 9, 0,  0, 0, 0],

    [9, 0, 2,  0, 5, 1,  3, 7, 0],
    [3, 0, 0,  9, 8, 0,  0, 0, 0],
    [0, 0, 5,  7, 0, 6,  0, 0, 0],

    [4, 0, 6,  0, 7, 5,  0, 3, 2],
    [0, 1, 0,  0, 0, 0,  7, 0, 5],
    [0, 0, 3,  0, 0, 0,  1, 9, 6]
]
```

Sudoku-oplosser in Python

- Stap 2: Strategie

- Probeer alle tactieken tot de puzzel is opgelost

```
def puzzel_opgelost():
    global puzzel
    for rij in puzzel:
        for getal in rij:
            if getal == 0:
                return False
    return True


while not puzzel_opgelost():
    probeer_voor_de_hand_liggend()
    probeer_geblokkeerd()
    probeer_hints()
    ...
print(puzzel)
```

Sudoku-oplosser in Python

- Stap 3: Tactieken implementeren
- Tactiek 1: Voor de hand liggende oplossingen
  - Er mist maar één getal in een rij, kolom, of blok
  - Dan kun je dat getal direct invullen
- Hiernaast: voor rijen

```
def probeer_voor_de_hand_liggend():
    global puzzel
    for rij in puzzel:
        ontbrekend = []
        for getal in range(1, 10):
            if not getal in rij:
                ontbrekend.append(getal)
        if len(ontbrekend) == 1:
            for i in range(9):
                if rij[i] == 0:
                    rij[i] = ontbrekend[0]
```

Stand van zaken

- Eenvoudigste tactiek *deels* geïmplementeerd
- 46 regels code
- Nog 37 tactieken te gaan!
- Oplosser waar dit op is gebaseerd: 3.000 regels JavaScript…
- **Kan dit makkelijker?**

```
puzzel = [
    [5, 0, 1,  6, 0, 7,  9, 0, 0],
    [0, 0, 9,  0, 0, 3,  2, 5, 0],
    [8, 2, 7,  0, 9, 0,  0, 0, 0],

    [9, 0, 2,  0, 5, 1,  3, 7, 0],
    [3, 0, 0,  9, 8, 0,  0, 0, 0],
    [0, 0, 5,  7, 0, 6,  0, 0, 0],

    [4, 0, 6,  0, 7, 5,  0, 3, 2],
    [0, 1, 0,  0, 0, 0,  7, 0, 5],
    [0, 0, 3,  0, 0, 0,  1, 9, 6]
]

def puzzel_opgelost():
    global puzzel
    for rij in puzzel:
        for getal in rij:
            if getal == 0:
                return False
    return True
```

```
def probeer_voor_de_hand_liggend():
    global puzzel
    for rij in puzzel:
        ontbrekend = []
        for getal in range(1, 10):
            if not getal in rij:
                ontbrekend.append(getal)
        if len(ontbrekend) == 1:
            for i in range(9):
                if rij[i] == 0:
                    rij[i] = ontbrekend[0]

def probeer_geblokkeerd():
    pass

def probeer_hints():
    pass

while not puzzel_opgelost():
    probeer_voor_de_hand_liggend()
    probeer_geblokkeerd()
    probeer_hints()

print(puzzel)
```

FUNDAMENT INFORMATICA

Sudoku-oplosser in Prolog

```
1  :- use_module(library(clpfd)).
2
3  sudoku(Rows) :-
4          length(Rows, 9), maplist(same_length(Rows), Rows),
5          append(Rows, Vs), Vs ins 1..9,
6          maplist(all_distinct, Rows),
7          transpose(Rows, Columns),
8          maplist(all_distinct, Columns),
9          Rows = [A,B,C,D,E,F,G,H,I],
10         blocks(A, B, C), blocks(D, E, F), blocks(G, H, I),
11       maplist(label, Rows).
12
13 blocks([], [], []).
14 blocks([A,B,C|Bs1], [D,E,F|Bs2], [G,H,I|Bs3]) :-
15         all_distinct([A,B,C,D,E,F,G,H,I]),
16         blocks(Bs1, Bs2, Bs3).
```

Sudoku-oplosser in Prolog

```
puzzel([
    [5, _, 1,  6, _, 7,  9, _, _],
    [_, _, 9,  _, _, 3,  2, 5, _],
    [8, 2, 7,  _, 9, _,  _, _, _],

    [9, _, 2,  _, 5, 1,  3, 7, _],
    [3, _, _,  9, 8, _,  _, _, _],
    [_, _, 5,  7, _, 6,  _, _, _],

    [4, _, 6,  _, 7, 5,  _, 3, 2],
    [_, 1, _,  _, _, _,  7, _, 5],
    [_, _, 3,  _, _, _,  1, 9, 6]
]).
```

```
?- puzzel(Puzzel), sudoku(Puzzel).
```

Puzzel =

5	3	1	6	2	7	9	8	4
6	4	9	8	1	3	2	5	7
8	2	7	5	9	4	6	1	3
9	6	2	4	5	1	3	7	8
3	7	4	9	8	2	5	6	1
1	8	5	7	3	6	4	2	9
4	9	6	1	7	5	8	3	2
2	1	8	3	6	9	7	4	5
7	5	3	2	4	8	1	9	6

Programmeerparadigma's

Puzzelen met Prolog

FUNDAMENT INFORMATICA

Programmeerparadigma's

- Een **programmeerparadigma** is een *manier* van programmeren

- Bij Fundament behandelen we er drie:
  - Imperatief
  - Objectgeoriënteerd
  - Logisch
- SLO heeft een keuzethema geschreven over het **functionele paradigma**
- Prolog gebruikt het **logische paradigma**

Paradigma's versus talen

- Een programmeerparadigma is niet een programmeertaal

- Voorbeeld: de Fibonacci-reeks: 0, 1, 1, 2, 3, 5, 8, 13
- De eerste twee getallen uit de reeks zijn 0 en 1
- Het volgende getal is de vorige twee bij elkaar opgeteld

#	Getal	Waarom
1	**0**	Gegeven
2	**1**	Gegeven
3	**1**	= **0** + **1**
4	**2**	= **1** + **1**
5	**3**	= **1** + **2**
6	**5**	= **2** + **3**
7	**8**	= **3** + **5**
8	**13**	= **5** + **8**

Paradigma's versus talen

- Fibonacci in één paradigma, maar verschillende talen

Python

```python
def fib(n):
    n1 = 0
    n2 = 1
    for i in range(n):

        n3 = n1 + n2
        n1 = n2
        n2 = n3

    return n2
```

PHP

```php
function fib($n) {
    $n1 = 0;
    $n2 = 1;
    for ($i = 0; $i < $n; $i++)
    {
        $n3 = $n1 + $n2;
        $n1 = $n2;
        $n2 = $n3;
    }
    return $n2;
}
```

JavaScript

```javascript
function fib(n) {
    let n1 = 0;
    let n2 = 1;
    for (let i = 0; i < n; i++)
    {
        n3 = n1 + n2;
        n1 = n2;
        n2 = n3;
    }
    return n2;
}
```

Paradigma's versus talen

- Fibonacci in verschillende paradigma's en talen

Imperatief Python

```python
def fib(n):
    n1 = 0
    n2 = 1
    for i in range(n):
        n3 = n1 + n2
        n1 = n2
        n2 = n3
    return n2
```

Logisch SWI-Prolog

```
fib(0, 1) :- !.
fib(1, 1) :- !.
fib(N, X) :-
    N1 is N - 1,
    N2 is N - 2,
    fib(N1, F1),
    fib(N2, F2),
    X is F1 + F2.
```

Functioneel Haskell

```
fib 0 = 1
fib 1 = 1
fib n =
    (fib (n - 1)) +
    (fib (n - 2))
```

Imperatief

- Reeks **instructies** die op volgorde wordt uitgevoerd
- If-statements, for-loops en while-loops veranderen de volgorde

- Lijkt veel op hoe een computer werkt
- Eerste en ‘makkelijkste’ paradigma, daarom populair

Talen:	Python, C#, PHP, JavaScript (alles in domein D)
Toepassingen:	Bijna overal!
Lesstof:	Fundament Domein D

Objectgeoriënteerd

- Functionaliteit gebundeld in **klassen**
- Elke klasse beschrijft een bepaald soort object met data
- Onderlinge communicatie met **methoden**

Talen:	Java, C++, C#
Toepassingen:	Applicaties met veel data
Lesstof:	Keuzethema Fundament over C#

Functioneel

- Programmeren in pure functies:
  - Zelfde input? Dan altijd dezelfde output
  - Geen **side effects**

- Programma's zijn erg voorspelbaar, bugs schrijven moeilijker

Talen: Haskell, Elm

Toepassingen: Wiskunde en economie

Lesstof: Keuzethema SLO over Elm

Logisch

- Beschrijf het **probleem** en de **oplossing**
- Beschrijf niet *hoe* het probleem wordt opgelost
- De taal zoekt een oplossing met een **formeel bewijs**

Talen:	Prolog (*programmation en logique*)
Toepassingen:	Puzzels en wiskunde
Lesstof:	Keuzethema Fundament over Prolog

Spoedcursus Prolog

Puzzelen met Prolog

FUNDAMENT INFORMATICA

Snelcursus Prolog: Feiten

- Een **feit** is een eigenschap die altijd waar is
- Het feit geldt voor één of meerdere argumenten

- Syntaxis: `eigenschap(argument1, argument2, ...).`

- Voorbeelden:
  - `vrouw(beatrix).`
  - `moeder(beatrix, willem_alexander).`
  - `vader(willem_alexander, alexia).`

Snelcursus Prolog: Regels

- Een **regel** is een feit dat onder voorwaarden waar is
- Syntaxis: **`head :- body.`**
- Lees: **`head`** is waar als **`body`** waar is

- Voorbeelden:
  - `ouder(Ouder, Kind) :- vader(Ouder, Kind); moeder(Ouder, Kind).`
  - `opa(Opa, Kleinkind) :- vader(Opa, Ouder), ouder(Ouder, Kleinkind).`

  `;` or (of) `,` and (en)
- **Variabelen** schrijf je met een hoofdletter
  - In dezelfde regel, met dezelfde naam, hebben ze dezelfde waarde

Snelcursus Prolog: Query's

- Prolog-programma's: verzameling van **feiten** en **regels**

- We voeren het programma niet uit, maar stellen een query:
  - Geldt `vader` voor `willem_alexander` en `alexia`?
    ```
    ?- vader(willem_alexander, alexia).
    ```
    True.
  - Voor welk **kind** geldt `ouder` voor `willem_alexander` en **dat kind**?
    ```
    ?- ouder(willem_alexander, Kind).
    ```
    Kind = **alexia**;
    Kind = **amalia**;
    Kind = **ariane**;

Snelcursus Prolog: Prolog en de realiteit

- Een argument, feit, of regel hoef je niet vooraf te **declareren**
- De invulling van je programma bepaal je zelf

- Prolog heeft dus geen flauw idee wat een `willem_alexander`, een `moeder` of een `beatrix` is!
- De programmeur geeft **betekenis** aan de verschillende symbolen

- Valkuil: logische fouten in de vertaalslag tussen Prolog en de **realiteit**

En nu zelf…

- https://bit.ly/koninklijkhuis
- Je gaat de volgende regels schrijven:
  - `oma(Oma, Kleinkind)`
  - `kind(Kind, Ouder)`
  - `broer_zus(Kind1, Kind2)`
- Heb je het onder de knie?
  - `oom_of_tante(OomOfTante, NeefOfNicht)`
    `OomOfTante` is de oom of tante van `NeefOfNicht`
  - `afstammeling(Afstammeling, Voorouder)`
    `Afstammeling` is het kind van (het kind van… enz.) `Voorouder`

Prolog in de klas

Puzzelen met Prolog

FUNDAMENT INFORMATICA

Prolog in de klas

- Het is niet eng, het is leuk!
- De module is getest in een klas van Adriaan – met groot succes
- Leuke uitdaging voor de *nerds*

  En dat is geen scheldwoord :-)

Leermomenten Prolog in de klas

- Het ‘kwartje’ valt sneller met *minder* programmeerervaring
- Ondersteuning, dus affiniteit van de docent, is nodig
- De basis van Prolog is snel onder de knie
- **Durf jij de uitdaging aan?**

Afsluiting

- Prolog-materiaal beschikbaar in Fundament Informatica
- Thuis verder aan de slag? https://bit.ly/koninklijkhuis
- Vragen over Prolog? wouter@instruct.nl
- Geïnteresseerd in Fundament? fundament@instruct.nl
- Dank voor de aandacht!