

Inleiding tot Programmeren

Examen Theorie en Praktijk

*1Ba Wiskunde, 3Ba Chemie
Academiejaar 2025-2026*

Toon Calders
toon.calders@uantwerpen.be

Sam Pieters
sam.pieters@uantwerpen.be

3 september 2025

Belangrijke informatie

- Het examen is gesloten boek en bestaat uit een deel theorie en een praktisch deel. Het gebruik van communicatiemiddelen zoals smartphones en smartwatches is niet toegestaan.
- Het internet mag enkel gebruikt worden voor Inginious. Via Inginious worden ook de slides en code fragmenten van de theorie (01. Inleiding loops en variabelen tot 07. Klassen 2 en getalrepresentaties) en cursusnotities voor het eerste deel van de cursus (Python, notebooks 00. Inleiding tot 16b. Objecten zijn mutable) beschikbaar gesteld. Ook een pdf van het boekje “Think Python. How to Think Like a Computer Scientist” is beschikbaar.
- Je hebt **3u** tijd om beide examens te maken. Het praktijkexamen moet op PC worden gemaakt en ingediend, het theorie-examen mag op PC, op papier, of gedeeltelijk op papier en op PC gemaakt worden. Duid wel duidelijk aan welke vragen op papier en welke vragen op PC werden ingediend.
- Dien je praktijk-oplossing in via Inginious. Zorg ervoor dat jouw inzending correct gebeurd is door ze te downloaden en te controleren.
- Lees een vraag eerst volledig alvorens ze op te lossen; **op het einde van elke vraag staan een aantal opmerkingen die je helpen om de vraag op te lossen.**

Theorie (Op 10)

Vraag 1 (3 punten)

```
1 def payable_with_wallet(bill, wallet):
2     if bill == 0:
3         return True
4     if bill < 0:
5         return False
6     if len(wallet) == 0:
7         return False
8     coin = wallet[0]
9     rest = wallet[1:]
10
11     payable_with_coin = payable_with_wallet(bill - coin, rest)
12     payable_without_coin = payable_with_wallet(bill, rest)
13
14     return payable_with_coin or payable_without_coin
```

Dit stukje code gaat na of je een bedrag `bill` exact kan betalen met munten en briefjes in jouw portefeuille, voorgesteld als een lijst `wallet` met de waardes van de munten en briefjes. Bijvoorbeeld: `payable_with_wallet(113, [50,20,20,10,5,5,2,2,1,1])` geeft `True`, terwijl `payable_with_wallet(113, [50,50,20,20,20,10,10,2,2,2,2])` de waarde `False` geeft (som van even getallen is steeds even en kan dus nooit 113 zijn).

- Leg de werking van deze recursieve functie uit en illustreer met een klein voorbeeldje.
- Bepaal de complexiteit van dit stukje code. Kies in functie van welke parameters je de complexiteit bespreekt en gebruik de $\mathcal{O}$ -notatie.
- Je mag ervan uitgaan dat alle aritmetische basisoperaties (+, *, ...) en vergelijkingen tussen getallen ($\leq$) in constante tijd uitgevoerd kunnen worden.

Vraag 2 (3 punten)

- (a) Leg uit wat een *hashtable* is.
- (b) Waarom is deze datastructuur geschikter om een *set* op te slaan in het geheugen dan een lijst?
- (c) Geef een voorbeeld van een stukje code dat een *set* gebruikt en waarvoor de implementatie met een hashtable efficiënter is dan met een lijst. Geef de computationele complexiteit in $\mathcal{O}$ -notatie van jouw stukje code voor beide gevallen (implementatie als een lijst versus implementatie met een hashtable).

Vraag 3: Getalvoorstellingen (2 punten)

Bekijk volgend programmaatje:

```
1 for i in range(1,11):
2     x=1/i
3     som=0.0
4     for j in range(i):
5         som+=x
6     print(i, som)
```

De uitvoer van dit programma is als volgt:

```
1 1.0
2 1.0
3 1.0
4 1.0
5 1.0
6 0.9999999999999999
7 0.9999999999999998
8 1.0
9 1.0000000000000002
10 0.9999999999999999
```

Verklaar deze output; waarom is het resultaat niet gewoon:

```
1 1.0
2 1.0
3 1.0
4 1.0
5 1.0
6 1.0
7 1.0
8 1.0
9 1.0
10 1.0
```

Leg ook uit waarom het resultaat soms groter is dan 1.0 en soms kleiner.

Vraag 4: Organisatie van het geheugen (2 punten)

Geef in detail alle objecten, hun type en hun structuur in het geheugen weer op het einde van volgend programma. Geef voor elke variabele aan naar welk object in het geheugen die verwijst en hoe dit object opgebouwd is.

```
1 from copy import copy, deepcopy
2
3 rij=[0]*3
4 M1=[rij]*3
5 M2=copy(M1)
6 M3=deepcopy(M2)
7 M1[0][0]=1
```

Praktijk (10 punten)

In dit praktijkexamen ga je een **doolhof** implementeren. Je stapt in de rol van programmeur die verantwoordelijk is voor het ontwerpen en bouwen van een nieuw computerspel waarbij een speler door dit doolhof kan bewegen.

De examenvragen zijn zo opgesteld dat ze **zo veel mogelijk onafhankelijk** zijn van elkaar. Als je vastloopt op een onderdeel, blijf daar dan niet te lang op hangen. Ga gerust verder met een volgende vraag, je kunt altijd later terugkomen op eerdere onderdelen.

Er zijn **4 vragen**. Bij elke vraag wordt het aantal punten gegeven die op die vraag staan.

Naast de correctheid van de implementatie wordt **ook de programmeerstijl beoordeeld**. Een uitzonderlijk goede of uitzonderlijk slechte programmeerstijl kan het resultaat tot 20% beïnvloeden. Met programmeerstijl wordt onder andere bedoeld: duidelijke code (variabelen hebben duidelijke namen, er werd commentaar in de code voorzien, de code is niet nodeloos complex), gebruik van datastructuren aangepast aan het probleem (bijvoorbeeld geen lijsten om sets op te slaan) en in mindere mate de efficiëntie van de code.

Vraag 1: Doolhof inlezen (2 punten)

Je krijgt een tekstbestand (.txt-bestand) dat een doolhof voorstelt. Elke regel in het bestand stelt een rij in het doolhof voor. Het doolhof bevat de volgende karakters:

- 'S': het startpunt
- 'F': het eindpunt
- '#': een muur (de speler kan hier niet doorheen)

Een voorbeeld van een doolhof vind je in "kleine-doolhof.txt":

```
DOOLHOF INFO
Dimensies: 8x7
```

```
DOOLHOF
#####
#S#   #
# # # #
# # # #
#     #
### # #
#     F#
#####
```

Schrijf een functie `lees_doolhof(bestandsnaam)` die een tekstbestand (.txt) inleest en omzet in een lijst van lijsten met karakters (`list[list[str]]`). Check ook of de dimensies correct zijn. Als de dimensies (lengte en/of breedte) niet gegeven zijn, wordt het volgende geprint in de console:

```
Dimensies niet gespecificeerd.
```

Als de dimensies wél gegeven zijn, maar komen niet overeen met de gegeven matrix, dan print je het volgende af:

```
Aantal rijen klopt niet.  Verwacht:  8,  Gevonden:  4.
```

Print hetzelfde voor de kolommen.

Voorbeeld

Volgende code genereert onderstaande output:

```
1 bestand = 'kleine_doolhof.txt'
2 doolhof = lees_doolhof(bestand)
3 print(doolhof)

[['#', '#', '#', '#', '#', '#', '#'],
 ['#', 'S', '#', ' ', ' ', ' ', ' ', '#'],
 ['#', ' ', '#', ' ', '#', ' ', '#'],
 ['#', ' ', '#', ' ', '#', ' ', '#'],
 ['#', ' ', ' ', ' ', ' ', ' ', '#'],
 ['#', '#', '#', ' ', '#', ' ', '#'],
 ['#', ' ', ' ', ' ', ' ', ' ', 'F', '#'],
 ['#', '#', '#', '#', '#', '#', '#]]
```

Opmerkingen

- Je kan ervan uitgaan dat de karakters voor startpunt, eindpunt, muren en paden altijd consistent zijn.
- De dimensies zijn niet gegeven in de volgende gevallen:
 - Het dimensieveld is leeg: `Dimensies:`
 - Een van de dimensies ontbreekt: `Dimensies: x7` of `Dimensies: 8x`

Andere gevallen, zoals een letter in plaats van een cijfer, hoeven niet te worden behandeld.

- Afgezien van de genoemde situaties hoef je geen rekening te houden met andere speciale gevallen.

Vraag 2: Doolhof logica (3 punten)**(a) Posities vinden (1 punt)**

Schrijf een functie `vind_posities(doolhof, teken)` die de een lijst van coördinaten (rij, kolom) geeft van een bepaald teken, zoals 'S', 'F', '#' of '-'. Zorg dat de lijst is gesorteerd. Eerst op rij index, als twee coördinaten zich op dezelfde rij bevinden dan sorteer je op kolom index.

Voorbeeld

```

1 doolhof = [['#', '#', '#', '#', '#', '#', '#'],
2           ['#', 'S', '#', '-', '-', '-', '#'],
3           ['#', '-', '#', '-', '-', '#', '-', '#'],
4           ['#', '-', '#', '-', '-', '#', '-', '#'],
5           ['#', '-', '-', '-', '-', '-', '-', '#'],
6           ['#', '#', '#', '-', '-', '#', '-', '#'],
7           ['#', '-', '-', '-', '-', '-', 'F', '#'],
8           ['#', '#', '#', '#', '#', '#', '#']]
9 start = vind_posities(doolhof, 'S')
10 print('startpositie:', start)
11 eind = vind_posities(doolhof, 'F')
12 print('eindpositie:', eind)
13 open_posities = vind_posities(doolhof, '-')
14 print('open_posities:', open_posities)

```

```
startpositie: [(1, 1)]
```

```
eindpositie: [(6, 5)]
```

```
open_posities: [(1, 3), (1, 4), (1, 5), (2, 1), (2, 3), (2, 5), (3, 1),
(3, 3), (3, 5), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (5, 3), (5, 5),
(6, 1), (6, 2), (6, 3), (6, 4)]
```

(b) Geldige buren (2 punten)

Schrijf een functie `geldige_buren(doolhof, positie)` die de posities teruggeeft waar je naartoe kan bewegen vanuit een bepaalde positie (dus niet door muren of buiten het doolhof).

Voorbeeld

```

1 doolhof = [['#', '#', '#', '#', '#', '#', '#'],
2           ['#', 'S', '#', '-', '-', '-', '#'],
3           ['#', '-', '#', '-', '-', '#', '-', '#'],
4           ['#', '-', '#', '-', '-', '#', '-', '#'],
5           ['#', '-', '-', '-', '-', '-', '-', '#'],
6           ['#', '#', '#', '-', '-', '#', '-', '#'],
7           ['#', '-', '-', '-', '-', '-', 'F', '#'],
8           ['#', '#', '#', '#', '#', '#', '#']]
9 start = (1, 1)
10 start_buren = geldige_buren(doolhof, start)
11 print('Buren-van-de-startpositie:', start_buren)
12 open_positie = (5, 5)
13 open_positie_buren = geldige_buren(doolhof, open_positie)
14 print('Buren-van-positie-(5,-5):', open_positie_buren)
15 muur_positie = (3, 4)
16 muur_positie_buren = geldige_buren(doolhof, muur_positie)
17 print('Buren-van-een-muur:', muur_positie_buren)

```

Buren van de startpositie: [(2, 1)]

Buren van positie (5, 5): [(4, 5), (6, 5)]

Buren van een muur: []

Opmerkingen

- Als de meegegeven positie een muur is, dan mag je [] terug geven ongeacht of de positie geldige burens heeft of niet.
- De eerste en laatste rij en kolom bestaan altijd uit muren.

Vraag 3: Pad logica (2 punten)**(a) Bestaat er een pad? (1 punt)**

Schrijf een functie `bestaat_pad(doolhof)` die bepaalt of er minstens één geldig pad bestaat van het startpunt 'S' naar het eindpunt 'F'. De functie moet `True` teruggeven als een dergelijk pad bestaat, en `False` als dit niet het geval is.

Voorbeeld

Volgende code genereert onderstaande output:

```

1 doolhof = [['#', '#', '#', '#', '#', '#', '#'],
2            ['#', 'S', '#', '-', '-', '-', '#'],
3            ['#', '-', '#', '-', '-', '#', '#'],
4            ['#', '-', '#', '-', '-', '#', '#'],
5            ['#', '-', '-', '-', '-', '-', '#'],
6            ['#', '#', '#', '-', '-', '#', '#'],
7            ['#', '-', '-', '-', '-', 'F', '#'],
8            ['#', '#', '#', '#', '#', '#', '#']]
9 if bestaat_pad(doolhof):
10     print("Er bestaat een pad van S naar F!")
11 else:
12     print("Er is geen pad van S naar F.")

```

Er bestaat een pad van S naar F!

Opmerkingen

- Een pad mag het doolhof niet verlaten. De eerste en laatste rij en kolom bestaan altijd uit muren, waardoor het pad binnen de grenzen van het doolhof blijft.
- Elk pad bestaat uit unieke coördinaten; een coördinaat mag nooit meer dan één keer in hetzelfde pad voorkomen.

(b) Doolhof met pad (1 punt)

Gegeven een lijst met coördinaten (rij, kolom) die een pad voorstellen, geef een nieuwe versie van het doolhof terug waarin het pad is weergegeven met het teken '.'.

De begin- en eindpunten, aangeduid met respectievelijk 'S' en 'F', moeten ongewijzigd blijven.

Let op: Het originele doolhof mag niet worden aangepast!

Voorbeeld

```

1 doolhof = [['#', '#', '#', '#', '#', '#', '#'],
2            ['#', 'S', '#', '-', '-', '-', '#'],
3            ['#', '-', '#', '-', '-', '#', '#'],
4            ['#', '-', '#', '-', '-', '#', '#'],
5            ['#', '-', '-', '-', '-', '-', '#'],
6            ['#', '#', '#', '-', '-', '#', '#'],
7            ['#', '-', '-', '-', '-', 'F', '#'],
8            ['#', '#', '#', '#', '#', '#', '#']]
9 pad = [
10     (1, 1), (2, 1), (3, 1), (4, 1), (4, 2),
11     (4, 3), (5, 3), (6, 3), (6, 4), (6, 5)
12 ]

```

```
13
14 nieuw = doolhof_met_pad(doolhof, pad)
15
16 print("Doolhof-met-pad:")
17 for rij in nieuw:
18     print("".join(rij))
19
20 print("\nOriginele-doolhof:")
21 for rij in doolhof:
22     print("".join(rij))
```

Doolhof met pad:

```
#####
#S#   #
#.# # #
#.# # #
#... #
###.# #
#   .F#
#####
```

Originele doolhof:

```
#####
#S#   #
# # # #
# # # #
#     #
### # #
#     F#
#####
```

Opmerkingen

- Een pad mag het doolhof niet verlaten. De eerste en laatste rij en kolom bestaan altijd uit muren, waardoor het pad binnen de grenzen van het doolhof blijft.
- Elk pad bestaat uit unieke coördinaten; een coördinaat mag nooit meer dan één keer in hetzelfde pad voorkomen.

Vraag 4: Speler klasse (3 punten)

Maak een klasse `Speler`, deze klasse wordt geïnitieerd met het doolhof. Deze klasse heeft drie variabelen: `positie`, `doolhof`, en `aantal_stappen`. De positie wordt geïnitieerd als de positie 'S' in het doolhof.

Verder bevat de klassen nog enkele methodes:

- `beweeg(self, richting)`: Beweegt de speler volgens de meegegeven richting op het doolhof en geef een boolean waarde terug als dit is gelukt of niet. Dit kan één van volgende strings zijn "links", "rechts", "boven" of "onder". Als een andere waarde wordt doorgegeven print je het volgende af:

"Ongeldige richting"

Als de speler door de richting tegen een muur botst, dan print je het volgende af:

"Botsing"

- `toon_status(self)`: Print de status van het doolhof en de speler. Print eerst het aantal gezette stappen. Print daarna het doolhof met de huidige positie van de speler. **Let op, de speler positie wordt geprint in het doolhof maar mag niet blijven staan. Het doolhof die wordt bewaard zou geen 'P' mogen bevatten.**
- `herstart(self)`: Zet de positie van de speler terug naar de begin positie. Zet het aantal stappen ook terug op 0.

Voorbeeld

Volgende code genereert onderstaande output:

```

1 doolhof = [
2   ['#', '#', '#', '#', '#'],
3   ['#', 'S', '-', '-', '#'],
4   ['#', '-', '#', '-', '#'],
5   ['#', '-', '-', 'F', '#'],
6   ['#', '#', '#', '#', '#'],
7 ]
8
9 speler = Speler(doolhof)
10 speler.toon_status()
11 speler.beweeg('rechts') # Gaat naar (1,2)
12 speler.toon_status()
13 speler.beweeg('omlaag') # Botsing, want muur
14 speler.toon_status()
15 speler.beweeg('rechts') # Gaat naar (1,3)
16 speler.toon_status()
17 speler.beweeg('omlaag') # Gaat naar (2,3)
18 speler.toon_status()
19 speler.beweeg('omlaag') # Gaat naar (3,3)
20 speler.toon_status()

```

Stappen gezet: 0

```

#####
#P #
# # #
# F#

```

#####

Stappen gezet: 1

#####

#SP #

#

F#

#####

Botsing

Stappen gezet: 1

#####

#SP #

#

F#

#####

Stappen gezet: 2

#####

#S P#

#

F#

#####

Stappen gezet: 3

#####

#S #

#P#

F#

#####

Stappen gezet: 4

#####

#S #

#

P#

#####