Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Tentamen:	Programmeren 1
Code:	NWI-NP033B
Docent:	H. Engelkamp
Telefoonnummer docent:	53367 / 0649848387
Datum:	17 januari 2019
Tijd:	8:30
Tentamenduur:	2 uur
Rekenmachine toegestaan:	Nee
Aantal vragen:	2
Beantwoord in:	Nederlands of Engels
Aantal pagina's (incl. deze):	5

Opgaven inleveren na afloop!

Het tentamen bestaat uit twee opgaven met elk vijf onderdelen. Bij elk onderdeel staat aangegeven hoeveel punten je kunt halen; 100 in totaal.

Schrijf leesbaar! Niet leesbaar = fout.

Start opgave 2 op een nieuw stuk papier. Voorzie alle vellen van je naam en studentnummer.

Je mag bij dit tentamen behalve schrijfgerei geen hulpmiddelen gebruiken.

1. Diverse vragen

Hieronder staan een aantal korte vragen. Geef op iedere vraag een beknopt, gemotiveerd antwoord.

1.1 (max. 10 punten) Harshadgetallen

Een Harshadgetal is een geheel getal dat door de som van zijn cijfers kan worden gedeeld. Zo is 21 deelbaar door 2 + 1 = 3. Beschouw de volgende code, die True of False zou moeten teruggeven als `n` respectievelijk wel of niet een Harshadgetal is:

```
def digit_sum(n):
    # Vind de som van de cijfers van het gehele getal n
    s_digits = list(str(n))
    dsum = 0
    for s_digit in s_digits:
        dsum += int(s_digit)

def is_harshad(n):
    return n % digit_sum(n) == 0
```

De volgende foutmelding verschijnt als je de functie aanroept:

```
>>> is_harshad(21)
TypeError: unsupported operand type(s) for %: 'int' and 'NoneType'
```

Leg duidelijk uit wat hier fout gaat en hoe het verbeterd kan worden.

1.2 (max. 10 punten) Complexiteit

Wat is de orde van de tijdscomplexiteitsgraad in n van de functie `digit_sum(n)` in opgave 1.1?

1.3 (max. 10 punten)

Verklaar de output van de volgende (interactieve) code:

```
>>> d = 8
>>> e = 2
>>> from math import *
>>> sqrt(d ** e)
16.88210319127114
```

N.B. sqrt() is de wortelfunctie uit de math module.

1.4 (max. 10 punten)

Wat is de waarde van de variabelen a, b, c, d en e na uitvoer van het volgende pythonscript?

```
a = 1 < 2 or 4 > 2
b = not 1 < 2 or 4 > 2
c = not (1 < 2 or 4 > 2)
d = not 0 < 1
e = 4 > 2 or 10/0 == 0
```

1.5 (max. 10 punten)

Wat is de output van het volgende programma? Leg duidelijk uit.

```
def pascal(n):
    if n == 1:
        return [1]
    else:
        previous_line = pascal(n-1)
        line = [1]
        for i in range(len(previous_line)-1):
            line += [previous_line[i] + previous_line[i+1]]
        line += [1]
    return line

print(pascal(5))
```

2. Programmeeropdracht: het vermoeden van Collatz.

Het vermoeden van Collatz wordt wel het eenvoudigste onopgeloste wiskunderaadsel genoemd. Het is het vermoeden dat de zogenaamde “hagelsteenrijen” uiteindelijk altijd uitlopen op het getal 1, welke beginwaarde er ook gekozen wordt. Zo’n hagelsteenrij ontstaat door te starten met een geheel getal groter dan nul, en vervolgens herhaaldelijk de volgende regels toe te passen: na een even getal is het volgende getal in de rij de helft van dat getal; na een oneven getal wordt het volgende getal verkregen door het met 3 te vermenigvuldigen en er 1 bij op te tellen. Aangekomen bij 1 eindigt de rij. Als je bijvoorbeeld begint met het getal 9 krijg je achtereenvolgens de volgende getallen:

9 - 28 - 14 - 7 - 22 - 11 - 34 - 17 - 52 - 26 - 13 - 40 - 20 - 10 - 5 - 16 - 8 - 4 - 2 - 1

De getallen in deze rij worden hagelsteengetallen genoemd omdat ze grillig omhoog- en omlaaggaan, net als hagelstenen in een wolk. Tot nu toe heeft nog niemand het vermoeden van Collatz kunnen bewijzen, maar er is ook nog geen enkel tegenvoorbeeld bekend. Inmiddels zijn alle begingetallen tot 10^{20} gecheckt.

In deze opdracht mag je ervan uitgaan dat er oplossingen bestaan voor de functies uit eerdere onderdelen, en ze toepassen. Behalve bij 2.2 mag je er ook telkens van uitgaan dat de functies correct, met de juiste argumenten, zullen worden aangeroepen.

2.1 (max 10 punten)

Schrijf de functie `volgend_hagelsteengetal(n)` die voor een getal `n` het eerstvolgende getal na `n` in de rij teruggeeft.

2.2 (max. 10 punten)

Schrijf de functie `hagelsteenrij(n)` die voor een startgetal `n` een pythonlist teruggeeft met daarin de gehele rij. Als deze functie wordt aangeroepen met een argument dat *geen* geheel getal groter dan of gelijk aan 1 is moet een lege lijst worden teruggeven. Je mag er in dit onderdeel van uitgaan dat het vermoeden van Collatz juist is.

2.3 (max. 10 punten)

De lengte van een rij en het hoogste getal dat tussendoor bereikt wordt zijn interessante eigenschappen. De rij die met 27 begint bestaat bijvoorbeeld uit maar liefst 112 getallen, en bereikt onderweg 9 232. De rij vanaf 9 663 bereikt zelfs 27 114 424!

Schrijf de functie `rijeigenschappen(n)` die een tabel naar het scherm schrijft met daarin de lengte van de rij en het hoogste tussenliggende getal voor alle hagelsteenrijen beginnend met de getallen 1 tot en met `n`. De opmaak van de tabel is hier niet belangrijk.

2.4 (max. 10 punten)

Als je de functie `hagelsteenrij(n)` zou aanroepen met een getal `n` dat onverhoopt toch een tegenvoorbeeld is van het vermoeden, dan loopt het geheugen vol; bijvoorbeeld doordat er een herhaling ontstaat.[1] Zo'n herhaling zou ook ontstaan als je niet zou stoppen bij 1: Je krijgt dan de oneindige lus 1 - 4 - 2 - 1 - 4 - 2 - 1 … .

Schrijf de functie `voldoet_aan_Collatz(n)` die `True` teruggeeft als het startgetal `n` voldoet aan het vermoeden, en `False` als `n` een tegenvoorbeeld is omdat er herhaling optreedt.

2.5 (max. 10 punten)

Schrijf de functie `vind_tegenvoorbeeld(n)` die *vanaf* een bepaald getal `n` alle gehele getallen afgaat op zoek naar een tegenvoorbeeld van het vermoeden van Collatz. Als het vermoeden klopt, zal deze functie blijven doorlopen zolang het computergeheugen toereikend is. Als er een tegenvoorbeeld is moet dat natuurlijk triomfantelijk worden medegedeeld. Laat de functie tussendoor bij elk duizendtal, als levensteken, het huidige te testen getal weergeven naar het scherm. Leesbaarheid is in deze opdracht belangrijker dan efficiëntie.

[1] De andere mogelijkheid, namelijk dat de rij 'opblaast', dus steeds grotere waarden oplevert tot in het oneindige, mag je buiten beschouwing laten.