Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Tentamen:	Programmeren 1
Code:	NWI-NP033B
Docent:	H. Engelkamp
Telefoonnummer docent:	53367 / 0649848387
Datum:	23 januari 2017
Tijd:	12:30
Tentamenduur:	3 uur
Rekenmachine toegestaan:	Nee
Aantal vragen:	2
Beantwoord in:	Nederlands of Engels
Aantal pagina's:	7

Opgaven inleveren na afloop!

Het tentamen bestaat uit twee opgaven met diverse onderdelen. Bij elk onderdeel staat aangegeven hoeveel punten je kunt halen. In totaal kun je 100 punten halen.

Schrijf leesbaar! Niet leesbaar = fout.

Start opgave 2 op een nieuw vel. 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 vragen over algoritmen. Geef voor iedere vraag een beknopt, gemotiveerd antwoord.

1.1 (max. 6 punten)

Bestudeer de volgende functie f:

```
def f (a, b, c, d):
    if a:
        if b and c and not d:
            return 0
        elif b and d:
            return 1
        else:
            return 2
    else:
        if not c:
            return 3
        if not d:
            return 4
        return 6
    return 7
```

1.1.1

Geef van iedere onderstaande aanroep van de functie f aan welk resultaat opgeleverd wordt:

```
print (f (True, True, True, True))
print (f (True, True, True, False))
print (f (True, False, True, False))
print (f (False, True, False, False))
print (f (False, True, True, True))
```

1.1.2

Voor welke aanroepen van de functie `f` is het resultaat gelijk aan de waarde 7?

1.2 (max. 10 punten)

Bestudeer de volgende functie `f`:

```
def f(a, b):
    if b == 0:
        return 0
    return a + f(a, b - 1)
```

Wat is het resultaat van de volgende aanroep van `f`; motiveer duidelijk je antwoord.

```
print(f(3, 4))
```

1.3 (max. 14 punten)

Beschouw onderstaande regels Python-code (in willekeurige volgorde):

```
1. break
2. while n % f == 0:
3. print('De priemfactoren van', n ,'zijn:')
4. def factorize(n):
5. if n == 1:
6. factorize(99999999)
7. print (f, end=' ')
8. n /= f
9. for f in range(2, n + 1):
```

Het dient om het getal 99999999 te ontbinden in priemfactoren, en geeft het volgende resultaat:

```
De priemfactoren van 99999999 zijn:
3 3 11 73 101 137
```

Opdracht: Zet de programmaregels in de juiste volgorde, en geef hierbij ook duidelijk aan hoever steeds ingesprongen moet worden. Een antwoord zou er dus zo uit kunnen zien:

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

N.B. De operator % levert de rest na deling op. Meerdere volgorden zijn goed; kies er slechts één.

1.4 (max. 10 punten)

Beschouw onderstaande Python code:

```
def search(lst, element):
    lowest_index = 0
    highest_index = len(lst) + 1
    # len(lst) geeft het aantal elementen van lst terug

    while highest_index > lowest_index:
        midd_index = (highest_index + lowest_index) // 2
        # // is integer deling

        if element < lst[midd_index]:
            lowest_index = midd_index
        elif element > lst[midd_index]:
            highest_index = midd_index
        elif element = lst[midd_index]:
            return midd_index
        else:
            return -1

lijst = [1,2,3,4,5,6,9]

print(search(lijst, 6))
```

Deze code zou, middels een *binary search*, de index van het element met waarde 6 uit lijst moeten teruggeven, 5 in dit geval. Als het element niet gevonden wordt zou dat aangegeven moeten worden met het resultaat -1. Het programma werkt echter niet correct; er zitten meerdere fouten in.

Opdracht: Identificeer de fouten in dit programma, en geef aan hoe die herstelt kunnen worden.

2 Programmeeropdracht: Sudoku

Een Sudoku is een getallenpuzzel die bestaat uit een rooster van negen bij negen vakjes dat bovendien nog verder is opgedeeld in negen deelroosters van elk drie bij drie vakjes. Een aantal van deze vakjes zijn al voorzien van een getal; dit is de start-Sudoku. Deze getallen mogen niet veranderd worden door de puzzelaar. Figuur 1 (a) geeft een voorbeeld van een mogelijke Sudoku-puzzel.

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

Figuur 1 (a) De start-Sudoku

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

(b) De oplossing

In de Sudoku-puzzel moeten in de lege plekken de cijfers 1 t/m 9 worden ingevuld, en wel zodanig dat:

- op elke horizontale rij elk cijfer één keer voorkomt;
- in elke verticale kolom elk cijfer één keer voorkomt;
- in elk drie-bij-drie blok elk cijfer één keer voorkomt.

Figuur 1b toont de oplossing die aan deze eigenschappen voldoet. Een start-Sudoku is altijd zodanig dat de oplossing uniek is.

In deze opdracht gebruiken we 2D lijsten (list-of-lists) om sudoku's te representeren in Python. De start-Sudoku uit figuur 1a ziet er dan als volgt uit:

```
start_sudoku = [[7,9,0, 0,0,0, 3,0,0],
                [0,0,0, 0,0,6, 9,0,0],
                [8,0,0, 0,3,0, 0,7,6],
                [0,0,0, 0,0,5, 0,0,2],
                [0,0,5, 4,1,8, 7,0,0],
                [4,0,0, 7,0,0, 0,0,0],
                [6,1,0, 0,9,0, 0,0,8],
                [0,0,2, 3,0,0, 0,0,0],
                [0,0,9, 0,0,0, 0,5,4]]
```

2.1 Sudoku tonen (max. 15 punten)

Bij het tonen van een Sudoku-puzzel moet je onderscheid maken tussen velden die een gebruiker wel kan veranderen en velden die een gebruiker niet kan veranderen. Om een Sudoku-puzzel `s` goed te kunnen tonen, heb je dus ook zijn start-waarden nodig. We kiezen ervoor om velden die niet veranderd kunnen worden aan te duiden met een *-teken voor en achter de waarde van het veld zodat ze opvallen.

Schrijf de functie `print_Sudoku (s, start)` die de sudoku s rij voor rij, kolom voor kolom, afdrukt en elk veld (r is de rij-index, k is de kolom-index):

- als s[r][k] gelijk is aan 0: "__.__"
- als s[r][k] is waarde v (ongelijk aan 0) en start[r][k] is gelijk aan 0: "__v__ "
- als s[r][k] is waarde v (ongelijk aan 0) en start[r][k] is ook gelijk aan v: "_*v*_ "

"_" stelt hierbij een spatie voor. Voorbeeld: (dit is dezelfde start-Sudoku als op de vorige pagina)

```
s = [[7,9,6, 0,0,0, 3,0,0],          start  = [[7,9,0, 0,0,0, 3,0,0],
     [0,0,0, 0,0,6, 9,0,0],                    [0,0,0, 0,0,6, 9,0,0],
     [8,0,1, 2,3,0, 0,7,6],                    [8,0,0, 0,3,0, 0,7,6],
     [0,0,0, 0,0,5, 0,0,2],                    [0,0,0, 0,0,5, 0,0,2],
     [0,0,5, 4,1,8, 7,0,0],                    [0,0,5, 4,1,8, 7,0,0],
     [4,0,0, 7,0,0, 5,1,0],                    [4,0,0, 7,0,0, 0,0,0],
     [6,1,0, 0,9,0, 0,0,8],                    [6,1,0, 0,9,0, 0,0,8],
     [0,8,2, 3,0,0, 0,9,7],                    [0,0,2, 3,0,0, 0,0,0],
     [3,0,9, 0,0,0, 0,5,4]]                    [0,0,9, 0,0,0, 0,5,4]]
```

De aanroep `print_Sudoku (s, start)` levert dan de volgende uitvoer op:

```
 *7*  *9*   6    .    .    .   *3*   .    .
  .    .    .    .    .   *6*  *9*   .    .
 *8*   .    1    2   *3*   .    .   *7*  *6*
  .    .    .    .    .   *5*   .    .   *2*
  .    .   *5*  *4*  *1*  *8*  *7*   .    .
 *4*   .    .   *7*   .    .    5    1    .
 *6*  *1*   .    .   *9*   .    .    .   *8*
  .    8   *2*  *3*   .    .    .    9    7
  3    .   *9*   .    .    .    .   *5*  *4*
```

Tips:

`'Blah{0:1d}blah'.format(5)` levert `'Blah5blah'` op;

`print('Blah', end='')` voorkomt dat er automatisch naar een nieuwe regel gesprongen wordt.

2.2 Correctheid deel-Sudoku bepalen (max. 20 punten)

Om te bepalen of een Sudoku (tot nu toe) correct is ingevuld moeten de negen velden van de negen rijen, de negen kolommen en de negen 3x3-blokken gecontroleerd worden. We noemen een dergelijke rij, kolom en 3x3-blok een deel-Sudoku. Een deel-Sudoku is correct als de waarden 1 t/m 9 maximaal één keer voorkomen; de waarde 0 mag meerdere keren voorkomen, omdat er meerdere nog lege vakjes kunnen zijn. Een deel-Sudoku kun je identificeren aan de hand van de kleinste en grootste rij-index en de kleinste en grootste kolom-index. Bijvoorbeeld:

- De vierde rij heeft kleinste en grootste rij-index 3 en 3 en kleinste en grootste kolom-index 0 en 8.
- De tweede kolom heeft kleinste en grootste rij-index 0 en 8 en kleinste en grootste kolom-index 1 en 1.
- Het linker-onder 3x3 blok heeft kleinste en grootste rij-index 6 en 8 en kleinste en grootste kolom-index 0 en 2.

Schrijf de functie `deelsudoku_is_correct (s, kleinste_rij, grootste_rij, kleinste_kolom, grootste_kolom)` die deze test uitvoert en `False` of `True` teruggeeft voor een deel-Sudoku van `s`.

2.3 Correctheid gehele Sudoku bepalen (max. 15 punten)

Met behulp van de functie `deelsudoku_is_correct` kun je de functie maken die bepaalt of een Sudoku s in zijn geheel tot nu toe correct is: `sudoku_is_correct(s)` die weer `True` of `False` teruggeeft. Schrijf deze functie. Je mag ervan uitgaan dat een uitwerking voor vraag 2.2 bestaat.

2.4 Sudoku opgelost (max. 10 punten)

Een Sudoku is opgelost als hij correct is ingevuld en alle velden een waarde tussen 1 t/m 9 hebben (de waarde 0 komt dus niet meer voor; er zijn geen lege vakjes). Implementeer deze laatste test-functie die bepaalt of een Sudoku `s` is opgelost: `sudoku_is_opgelost (s)`. Het resultaat van deze functie is weer `True` of `False`. Je mag ervan uitgaan dat een uitwerking voor vraag 2.3 bestaat.

```python
## 1.1

def f (a, b, c, d):
    if a:
        if b and c and not d:
            return 0
        elif b and d:
            return 1
        else:
            return 2
    else:
        if not c:
            return 3
        if not d:
            return 4
        return 6
    return 7

#1.1.1
print (f (True, True, True, True))      # 1
print (f (True, True, True, False))     # 0
print (f (True, False, True, False))    # 2
print (f (False, True, False, False))   # 3
print (f (False, True, True, True))     # 6

# 1.1.2 Geen enkele aanroep kan 7 opleveren. Bij een if statement met een else
# clausule wordt minimaal één van de twee uitgevoerd. Return betekent: "ga terug", dus
# dat beeindigt altijd direct de functie. Meerdere waarden teruggeven kan wel
# maar dan moet je dat via een enkele structuur doen, bijvoorbeeld een list of
# een dictionary.


## 1.2
def f(a, b):
    if b == 0:
        return 0
    return a + f(a, b - 1)


print(f(3, 4))
# 12 (= b * a).
# f(3,4)=3 + f(3,3) = 3+(3+f(3,2)) = 3+(3+(3+f(3,1))) = 3+(3+(3+(3+f(3,0)))) =
# 3+(3+(3+(3+0))) = 12

## 1.3 Mogelijke correcte volgorden:
def factorize(n):                                   #4
    print('De priemfactoren van',n,'zijn:')         #3
    for f in range(2,n+1):                          #9
        while n % f == 0:                           #2
            print (f, end=' ')                      #7 (8)
            n/=f                                    #8 (7)
        if n==1:                                    #5
            break                                   #1
factorize(99999999)                                 #6
## 1.3
def factorize(n):                                   #4
    print('De priemfactoren van',n,'zijn:')         #3
    for f in range(2,n+1):                          #9
        if n==1:                                    #5
            break                                   #1
        while n % f == 0:                           #2
            print (f, end=' ')                      #7 (8)
            n/=f                                    #8 (7)
factorize(99999999)                                 #6


## 1.4 (met fouten)
def search(lst, element):
    lowest_index = 0
    highest_index = len(lst) + 1 # +1 gaat fout
```

```
    while highest_index > lowest_index:
        midd_index = (highest_index + lowest_index) // 2
        if element < lst[midd_index]: #<
            lowest_index = midd_index
        elif element > lst[midd_index]: #>
            highest_index = midd_index
        elif element = lst[midd_index]: # Altijd waar
            return midd_index
        else:
            return -1   # wordt nooit bereikt

                # Oneindige loop als element niet voorkomt

lijst = [1,2,3,4,5,6,9]
print(search(lijst, 6))

## 1.4 (mogelijk correct):

def search(lst, el):
    lowest_index = 0
    highest_index = len(lst)        # len(lst)-1 vindt laatste element nooit
    while highest_index > lowest_index:
        midd_index = (highest_index+lowest_index) // 2
        if el > lst[midd_index]:            # < veranderd in  >
            if lowest_index == midd_index:  # anders oneindige loop voor
                                            # gezochte elementen die niet in de
                                            # lijst voorkomen
                                            # el > lst[-1]
                return -1
            lowest_index = midd_index
        elif el < lst[midd_index]:          # > veranderd in <
            highest_index = midd_index
        else:                               # elif onnodig: niet kleiner of
                                            # groter betekent gelijk
            return midd_index

    return -1                       # alleen als gezochte element kleiner dan
                                    # lst[0]



lijst=[1,2,3,4,5,6,9]
print(search(lijst,6))

## 2

# benodigde Sudoku's genereren (niet nodig op het tentamen)

sudoku = [[7,9,6, 0,0,0, 3,0,0],
            [0,0,0, 0,0,6, 9,0,0],
            [8,0,1, 2,3,0, 0,7,6],
            [0,0,0, 0,0,5, 0,0,2],
            [0,0,5, 4,1,8, 7,0,0],
            [4,0,0, 7,0,0, 5,1,0],
            [6,1,0, 0,9,0, 0,0,8],
            [0,8,2, 3,0,0, 0,9,7],
            [3,0,9, 0,0,0, 0,5,4]]

start_sudoku = [[7,9,0, 0,0,0, 3,0,0],
            [0,0,0, 0,0,6, 9,0,0],
            [8,0,0, 0,3,0, 0,7,6],
            [0,0,0, 0,0,5, 0,0,2],
            [0,0,5, 4,1,8, 7,0,0],
            [4,0,0, 7,0,0, 0,0,0],
            [6,1,0, 0,9,0, 0,0,8],
            [0,0,2, 3,0,0, 0,0,0],
            [0,0,9, 0,0,0, 0,5,4]]

oplossing = [[7,9,6, 8,5,4, 3,2,1],
            [2,4,3, 1,7,6, 9,8,5],
            [8,5,1, 2,3,9, 4,7,6],
```

```
            [1,3,7, 9,6,5, 8,4,2],
            [9,2,5, 4,1,8, 7,6,3],
            [4,6,8, 7,2,3, 5,1,9],
            [6,1,4, 5,9,7, 2,3,8],
            [5,8,2, 3,4,1, 6,9,7],
            [3,7,9, 6,8,2, 1,5,4]]


# 2.1

def print_sudoku(s, start_s):
    for r in range(9):
        for k in range(9):
            if s[r][k]==0:
                print ('  .  ', end='')
            elif s[r][k]==start_s[r][k]:
                print(' *{0:1d}* '.format(s[r][k]), end='')
            else:
                print('  {0:1d}  '.format(s[r][k]), end='')
        print()

# 2.2 Ik maak een lijst "num_exists" waarin ik per index bijhoud of die al is
# voorgekomen of niet. Initieel moet die lijst 10 Falses bevatten. Ik "loop"
# over de gevraagde rijen en kolommen, en als een element is geweest wordt het
# corresponderende element True.
def deelsudoku_is_correct(s, kleinste_rij, grootste_rij, kleinste_kolom,
grootste_kolom):
    num_exists = [False]*10 # of [False, False, ... ,False]
    for r in range(kleinste_rij, grootste_rij+1):
        for k in range(kleinste_kolom, grootste_kolom+1):
            if s[r][k]>0: # 0 mag vaker voorkomen
                if num_exists[s[r][k]]: # Komt al in de lijst voor; mag niet!
                    return False
                num_exists[s[r][k]]=True
    return True

# 2.3 Ik roep hier de voorgaande functie op voor alle rijen, kolommen,
# en apart voor alle 3x3 blokken. Zodra er een niet klopt kan False
# teruggegeven worden. Als dat nergens gebeurt is het blijkbaar ok en kan
# True worden teruggegeven.
def sudoku_is_correct(s):
    # 3x3 deel-sudoku
    for r in range(0, 9, 3):
        for k in range(0, 9, 3):
            if not deelsudoku_is_correct(s, r, r+2, k, k+2):
                return False
    for rk in range(9):
        # kolommen
        if not deelsudoku_is_correct(s, 0, 8, rk, rk):
            return False
        # rijen
        if not deelsudoku_is_correct(s, rk, rk, 0, 8):
            return False
    return True

#2.4  Er mogen geen nullen meer voorkomen en de gehele Sudoku moet
# correct zijn in de betekenis van 2.3 en 2.2
def sudoku_is_opgelost(s):
    if sudoku_is_correct(s):
        for r in range(9):
            for k in range(9):
                if s[r][k]==0:
                    return False
        return True
    else:
        return False

# nog compacter:
def sudoku_is_opgelost(s):
    for r in range(9):
        for k in range(9):
```

```
            if s[r][k]==0:
                return False
    return sudoku_is_correct(s)


print_sudoku(sudoku, start_sudoku)
print(sudoku_is_correct(sudoku))
print(sudoku_is_opgelost(sudoku))
print(sudoku_is_opgelost(oplossing))
```