

Tentamen Programmeermethoden NA

Vrijdag 16 december, 14.00 - 17.00 uur

Universiteit Leiden — Informatica

Het tentamen bestaat uit 4 opgaven verdeeld over 3 pagina's. De duur van het tentamen is 3 uur. De te behalen punten (totaal 100) staan tussen haakjes bij de opgaven.

Bij alle functies moeten de variabelen (constanten eventueel uitgezonderd) als parameter of lokale variabele voorkomen. Gebruik `___` om één niveau van inspringen aan te duiden. Twee keer dit karakter betekent dus twee keer inspringen. Denk aan de dubbele punten!

Veel succes!

1. (35 punten) We hebben een lijst `A` met `len(A)` (gegarandeerd > 0 en een even getal) gehele getallen.

a. (4) Schrijf een Python-functie `evenveel(A)` die precies `True` teruggeeft als `A` evenveel positieve getallen (≥ 0) als negatieve (< 0) bevat.

b. (6) Schrijf een Python-functie `epo(negpos, A, start)` die de lijst-index teruggeeft van het eerste positieve (als `negpos True` is) of negatieve (als `negpos False` is) getal in `A` vanaf (en inclusief) positie `start`. Als zo'n getal niet voorkomt, moet de lengte van `A` worden geretourneerd.

c. (8) Neem aan dat `A` evenveel positieve getallen als negatieve bevat. We willen de positieve en negatieve getallen in `A` om en om krijgen, te beginnen met een positief getal. Schrijf een Python-functie `omenom(A)` die dit als volgt doet. Laat `i` vanaf 0 oplopen en gebruik de functie van `b` om eventueel een posi/negatief getal te zoeken, en ruil dit met `A[i]` om.

d. (4) Stel dat de positieve getallen in `A` aan het begin staan, en de negatieve aan het eind. Hoeveel vergelijkingen tussen getallen (tests of een getal negatief of positief is) doet het algoritme van `c` dan, uitgedrukt in `n`? (Deze worden ook gedaan in de aanroepen van `b`.)
Tip: probeer eens `n = 10`.

e. (5) Schrijf een Python-functie `controle(A)` die het resultaat van het algoritme van `c` controleert en `True` teruggeeft wanneer `A` aan de eisen voldoet en anders `False`. Stop direct zodra het resultaat bekend is.

f. (8) Schrijf een Python-functie `langste(A)` die de lengte uitrekent van een langste stijgende (of beter: niet-dalende) aangesloten deelrij in de lijst `A`. De functie geeft als returnwaarde zowel de lengte van de gevonden deelrij als het niet-afgeronde gemiddelde van de getallen in de betreffende deelrij. De functie wordt aangeroepen als volgt: `lengte, gem = langste(A)`. Voor de lijst `1 7 1 1 2 7 6 3` zou de returnwaarde `(4, 2.75)` zijn, namelijk de lengte van de deelrij `1 1 2 7` en `gem` via $11/4$. (Als er meer deelrijen dezelfde maximale lengte realiseren: het gemiddelde van de laatste).

2. (20 punten)

a. (4) Bij een functie kun je te maken hebben met *locale* en *globale* variabelen. Verder onderscheiden we *formele* en *actuele* parameters. Leg deze vier begrippen duidelijk uit.

b. (6) Gegeven een Python-programma met daarin de volgende twee functies:

```
def donald(x, y):
    iets = 0
    a = y
    while a > 0:
        iets += x
        y -= 1
        print "S", a, iets, y
        a -= 1
    return iets

def katrien(a, b):
    x = a - 1
    b += 1
    a -= 1
    a = donald(b, a) + donald(a, a)
    print x, a, b
    return a * (2 - x)
```

Verder zijn de globale variabelen `x`, en `y` gegeven (beide van type `int`). Wat is dan de uitvoer van het volgende stukje programma (leg je antwoord duidelijk uit):

```
x, y = 4, 2
print katrien(x, y), x, "en", y
```

c. (5) Geef een functie `dagobert(a,b)` die dezelfde returnwaarde oplevert als `katrien` voor alle mogelijke waarden van de parameters, maar die uit slechts één `return`-statement bestaat, en waarin `donald` niet meer wordt aangeroepen.

d. (5) Stel nu we plaatsen bovenaan `katrien` het statement `global x` en we draaien het stukje programma onder `b` opnieuw. Is de uitvoer nu anders? En hoe zit dat wanneer we alleen het `print`-statement (met de aanroep naar `katrien`) meerdere keren uitvoeren?

3. (10 punten) Gegeven een NumPy array([[20, 21, 22, 23, 24, 25, 26, 27],
array M zoals hiernaast weergegeven. [28, 29, 30, 31, 32, 33, 34, 35],
Schrijf voor de hieronder gegeven [36, 37, 38, 39, 40, 41, 42, 43],
slices van M de juiste expressie die [44, 45, 46, 47, 48, 49, 50, 51]])
resulteert in die slice.

a. (2) `array([24, 32, 40, 48])`

b. (2) `array([40, 41, 42, 43])`

c. (2) `array([[25, 26, 27],
[33, 34, 35],
[41, 42, 43],
[49, 50, 51]])`

d. (2) `array([[36, 38, 40, 42],
[44, 46, 48, 50]])`

e. (2) Geef de NumPy-expressie die een 1-dimensionale array oplevert met daarin de som voor elke *kolom* van M. Resultaat: `array([128, 132, 136, 140, 144, 148, 152, 156])`.

4. (35 punten) We hebben een n bij n (> 0) NumPy array S met gehele getallen ≥ 0 . Dit stelt hier een geheel of gedeeltelijk ingevulde *Sudoku* voor. Lege vakjes geven we aan met 0, de overige vakjes bevatten een getal tussen 1 en n . We bekijken de eenvoudige variant van Sudoku waarbij alleen in rijen en kolommen elk getal uit 1 tot en met n precies één keer moet voorkomen. (De gebruikelijke restricties voor blokken vervallen). Hiernaast staat een voorbeeld met $n = 5$. De variabele n is als globale “constante” gedefinieerd en hoeft bij deze opgave niet doorgegeven te worden als parameter.

0	0	4	2	0
3	2	0	0	5
1	4	3	5	2
0	0	2	0	0
4	3	0	1	0

a. (5) Schrijf een Python-functie `leeg(S)` die het aantal lege vakjes in S bepaalt en als returnwaarde oplevert. Voor het voorbeeld hierboven: 11.

b. (10) Schrijf een Python-functie `okee(S, i)` die controleert of elke positieve waarde 1 tot en met n precies één keer in rij i voorkomt. Zo ja, dan levert de functie `True` op, anders `False`. Gebruik hierbij eventueel een hulp-lijst of hulp-array. Voor het voorbeeld: `False` voor rij 0 en `True` voor rij 2.

c. (10) Schrijf een Python-functie `invulbaar(S, i, j)` die voor een gegeven vakje (i, j) in een tot dusver goed ingevulde Sudoku berekent hoeveel getallen daar volgens de regels kunnen worden ingevuld en de kleinste van die mogelijkheden teruggeeft. De functie wordt aangeroepen als `aantal, kleinste = invulbaar(S, i, j)`. Als er 0 mogelijkheden zijn, maak `kleinste` dan ook 0. In het voorbeeld kunnen voor vakje $(0, 1)$ twee getallen worden ingevuld, de kleinste is 1 en voor vakje $(4, 2)$ één getal, de kleinste is 5.

d. (10) We gaan nu proberen de oorspronkelijke Sudoku verder in te vullen. Schrijf hiertoe een functie `invullen(S)` die als volgt te werk gaat: zoek een vakje (maakt niet uit welk) waarvoor het aantal in te vullen waardes $\neq 0$ is. Gebruik hier `c`; vul de door `invulbaar` opgeleverde `kleinste`-waarde in. Herhaal dit totdat er geen lege vakjes meer zijn (de Sudoku is opgelost; retourneer `True`) of totdat er geen invulbare vakjes meer zijn (geef `False`).