Opdrachten week 3 les 1 – Lijsten in Python

In de vorige twee weken heb je geleerd om eenvoudige programmeerproblemen op een pc op te lossen met behulp van de programmeertaal Python en de IDE Thonny. Deze les ga je leren hoe je in Python lijsten kunt gebruiken om bij elkaar behorende variabelen te bundelen. Ook leer je deze les om de werking van een programma grafisch weer te geven door middel van een flowchart.

Lees nu hoofdstuk 10 tot en met paragraaf 10.3 van het boek[1].

3.1.1 Schrijf een functie genaamd avg die het gemiddelde van een rij getallen berekent en teruggeeft. Deze rij getallen wordt als argument, via een Python `list`, doorgegeven aan de functie. Test deze functie met de testcode uit listing 1. De gewenste uitvoer is gegeven in figuur 1.

```
test_list = [3.2, 5.7, 8.5, 9.1]
print(avg(test_list))
```

Listing 1: Testcode voor de functie avg, zie avg.py.

Figuur 1: De gewenste uitvoer van de testcode gegeven in listing 1.

3.1.2 Schrijf een functie genaamd `std_dev` die de standaardafwijking[2] (Engels: standard deviation) van een rij getallen berekent en teruggeeft. Deze rij getallen wordt als argument, via een Python `list`, doorgegeven aan de functie. Hoe je de standaardafwijking aangeduid met σ kunt berekenen is gegeven in vergelijking (1). Hierin is N het aantal getallen in de rij, x_i het i^{de} getal uit de rij en μ het gemiddelde van de rij. Het teken $\sum$

[1] Allen B. Downey. *Think Python: How to Think Like a Computer Scientist*. 2de ed. Green Tea Press, 2016. ISBN: 978-1-4919-3936-9. URL: http://greenteapress.com/wp/think-python-2e/.

[2] Als je niet weet wat de standaardafwijking van een rij getallen is, zie dan https://nl.wikipedia.org/wiki/Standaardafwijking.

is het sommatieteken[3]. Tip: je hebt zojuist al een functie geschreven die μ berekent, zie opdracht 3.1.1.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \quad \text{met} \quad \mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{1}$$

Test deze functie met de testcode uit listing 2. De gewenste uitvoer is gegeven in figuur 2.

```
test_list = [3.2, 5.7, 8.5, 9.1]
print(std_dev(test_list))
```

Listing 2: Testcode voor de functie avg, zie `std_dev.py`.

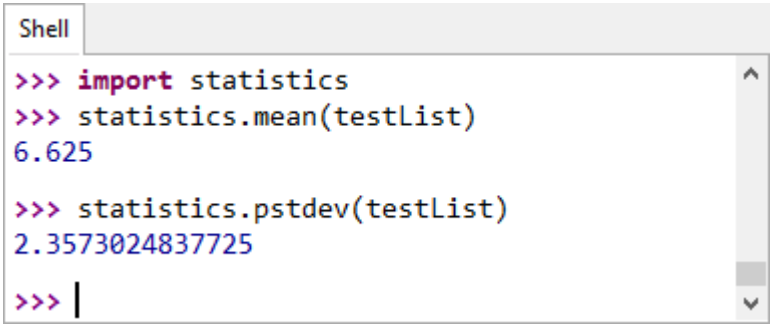
Figuur 2: De gewenste uitvoer van de testcode gegeven in listing 2.

3.1.3 Er is in Python een module `statistics`[4] beschikbaar die gebruikt kan worden om het gemiddelde en de standaardafwijking te berekenen. Zie figuur 3. Pas de testcodes gegeven in listing 1 en listing 2 zodanig aan dat de door jouw functies berekende waarde voor het gemiddelde respectievelijk de standaardafwijking vergeleken wordt met de door de functies uit de module `statistics` berekende waarde. Het aangepaste testprogramma moet `Gelijk` of `Niet gelijk` afdrukken afhankelijk van het resultaat van de vergelijking. Verklaar eventuele verschillen.

Als programma's ingewikkelder worden, is het handig om de werking van een programma visueel weer te kunnen geven. Een veel gebruikt diagram om de werking van een programma

[3] Zie eventueel: https://nl.wikipedia.org/wiki/Sommatie.

[4] Zie: https://docs.python.org/3/library/statistics.html.



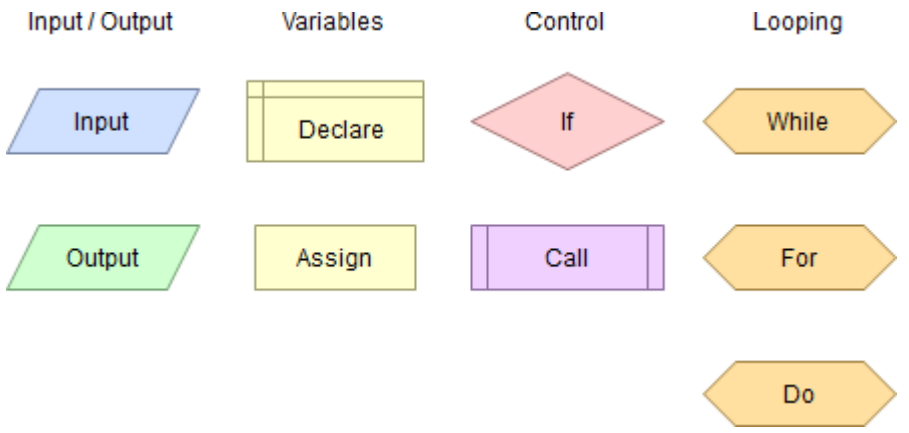
Figuur 3: Je kunt het gemiddelde en de standaardafwijking ook berekenen met de functies uit de module `statistics`.

weer te geven is het zogenoemde stroomdiagram (Engels: flowchart)[5]. Een flowchart bestaat uit een beperkt aantal symbolen die met elkaar worden verbonden via pijlen. Wij gebruiken het programma Flowgorithm om flowcharts te tekenen. In figuur 4 kun je de symbolen zien die je in een flowchart kunt gebruiken. De meeste symbolen spreken voor zich. Het symbool 'Declare' wordt gebruik om een variabele te declareren. In Python is het niet nodig om een variabele te declareren dus bij het visualiseren van een Python programma maak je geen gebruik van dit symbool. Volgende week zul je de programmeertaal C leren kennen. In die taal is het wel nodig om een variabele te declareren en dan komt dit symbool goed van pas. Het symbool 'Call' wordt gebruik om een functieaanroep mee weer te geven. Het symbool 'Do' heeft geen nut bij het visualiseren van een Python programma omdat deze programmeertaal geen **do**-statement bevat. In de programmeertaal C kun je wel een **do**-statement gebruiken en dan komt dit symbool goed van pas.

3.1.4 Open het programma Flowgorithm in de Liquit Workspace of installeer[6] het programma Flowgorithm op je eigen pc. Bij opdracht 2.2.8 heb je een recursieve functie geschreven om de grootste gemene deler van twee positieve gehele getallen te berekenen. Het hierbij gebruikte recursieve algoritme werd daarbij gedefinieerd zoals weergegeven in vergelijking (2).

[5] De manier waarop een flowchart getekend moet worden is vastgelegd in de internationale standaard ISO 5807:1985.

[6] Zie `http://www.flowgorithm.org/download/index.html`.



Figuur 4: De symbolen die je kunt gebruiken bij het tekenen van een flowchart met behulp van het programma Flowgorithm.

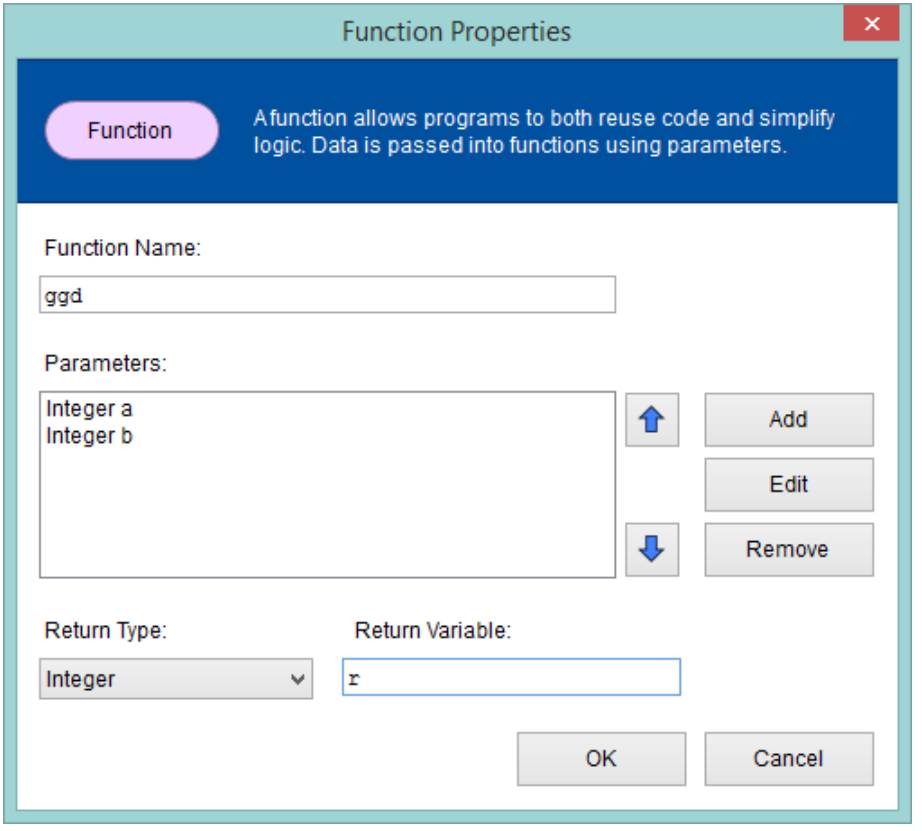
$$\text{ggd}(a,b) = \begin{cases} a & \text{als } a = b \\ \text{ggd}(a-b,b) & \text{als } a > b \\ \text{ggd}(a,b-a) & \text{als } a < b \end{cases} \tag{2}$$

Start het programma Flowgorithm en maak een nieuwe functie aan met de menuoptie Program » Add Function... . Er verschijnt een dialoogvenster waarin je de naam, parameters en returnvariabele van de functie in kunt voeren. Bij het invoeren van de parameters moet ook het type van de parameters worden opgegeven. In Python is dat niet nodig maar in Flowgorithm wel. Volgende week zul je de programmeertaal C leren kennen en in die taal moet het type van parameters wel worden opgegeven als je een functie definieert. In Flowgorithm kan een functie maar één return statement hebben. De expressie die dit returnstatement retourneert moet worden opgegeven in het dialoogvenster. Vul de naam, parameters en returnvariabele in voor de functie ggd, zie figuur 5.

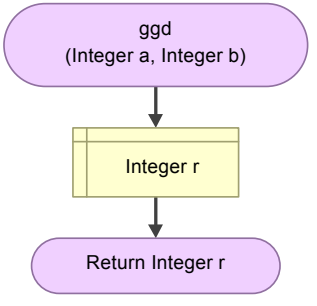
Als het goed is, verschijnt nu de flowchart gegeven in figuur 6.

Voeg nu de juiste symbolen ('If' en 'Assign') toe om de implementatie van het algoritme gegeven in vergelijking (2) te visualiseren[7]. Je kunt symbolen toevoegen door op de

[7] Een mogelijke oplossing vind je in figuur 7.

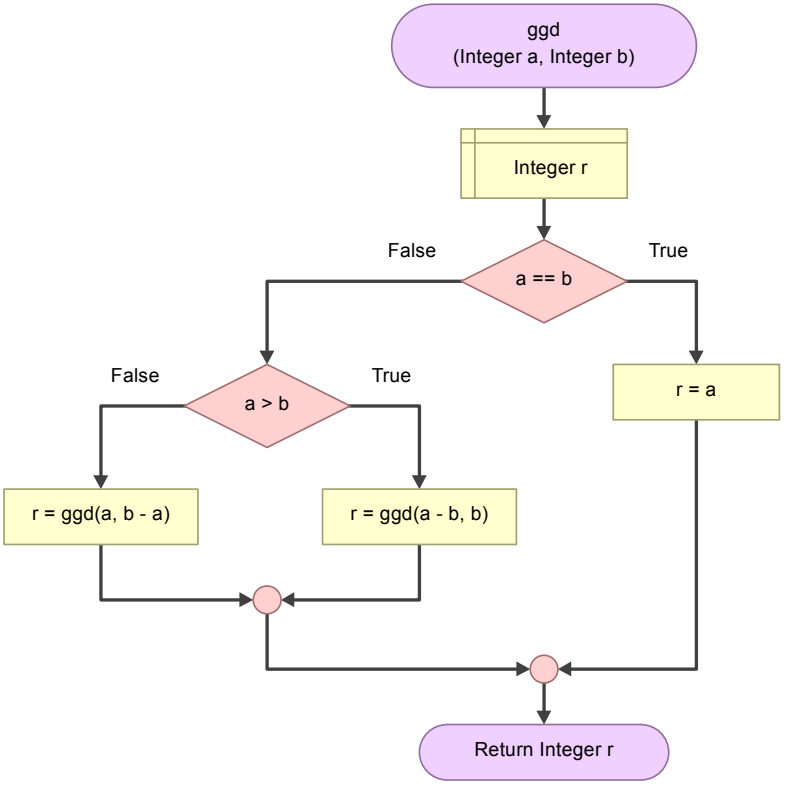


Figuur 5: De naam, parameters en returnvariabele van de functie ggd.



Figuur 6: Een flowchart van de functie ggd waarbij de functionaliteit nog niet is ingevuld.

pijl te klikken waar je een symbool wilt invoegen. Er verschijnt dan een popup window waarin je het gewenste symbool kunt selecteren.



Figuur 7: Een flowchart van de recursieve functie ggd.

Het programma Flowgorithm kan een flowchart omzetten naar programmacode voor verschillende programmeertalen waaronder Python.

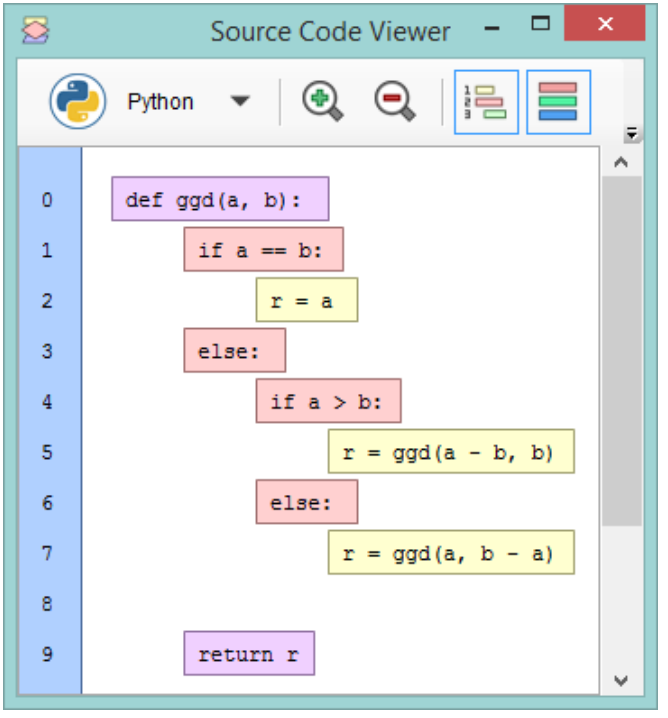
3.1.5

A Zet de flowchart die gegeven is in figuur 7 om naar code met behulp van de knop 'Source Code Viewer…'  en kies linksboven voor Python. Als het goed is, verschijnt de in figuur 8 gegeven Python code.

B Sla deze code op in een bestand met behulp van de knop 'Save'  .

C Test deze code in Thonny.

D Vergelijk de code die door Flowgorithm geproduceerd is met de code die je zelf hebt geschreven bij opdracht 2.2.8. Welke code vind je beter en waarom?



Figuur 8: De door Flowgorithm gegeneerde Python code vanuit het in figuur 7 gegeven flowchart.

3.1.6 Bij opdracht 2.2.12 heb je een iteratieve functie geschreven om de grootste gemene deler van twee positieve gehele getallen te berekenen. Het iteratieve algoritme dat je hebt gebruikt om de grootste gemene deler van twee positieve gehele getallen a en b te bepalen bestond uit de volgende stappen:

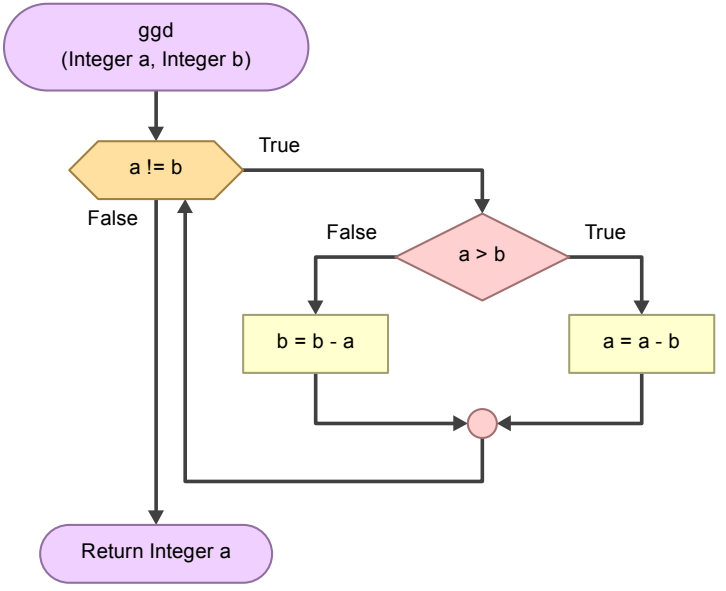
- herhaal de volgende code zolang $a \neq b$:
  - als $a > b$, dan $a = a - b$
  - anders $b = b - a$
- return a

A Teken met behulp van Flowgorithm een flowchart van deze functie[8].

B Genereer Python code voor de iteratieve functie ggd met behulp van Flowgorithm. Test deze code.

[8] Een mogelijke oplossing is gegeven in figuur 9.

C Vergelijk de code die door Flowgorithm geproduceerd is met de code die je zelf hebt geschreven bij opdracht 2.2.12. Welke code vind je beter en waarom?



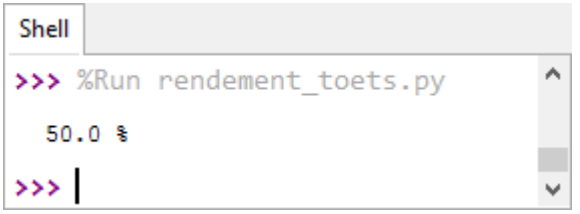
Figuur 9: Een flowchart van de iteratieve functie ggd.

3.1.7 Een docent heeft de resultaten van een toets opgeslagen in een rij genaamd `resultaten`. De toetsresultaten zijn getallen tussen 1.0 en 10.0 (inclusief) met één cijfer achter de decimale punt. Een manager eist van deze docent een rendement van minimaal 70%. Het rendement is gedefinieerd als het aantal voldoendes (toetsresultaat $\geq$ 5.5) gedeeld door het aantal toetsresultaten * 100%. Als minder dan 70% van de resultaten voldoende is, dan moeten alle resultaten worden opgehoogd met 0.1 net zolang tot het gewenste rendement van 70% is behaald.

A Schrijf een functie genaamd `rendement_toets` die het rendement van een toets berekent en teruggeeft. De rij met resultaten van de toets wordt als argument, via een Python `list`, doorgegeven aan de functie. Test deze functie met de testcode uit listing 3. De gewenste uitvoer is gegeven in figuur 10.

```
toetsresultaten = [1.2, 3.6, 9.2, 8.2, 3.5, 5.5, 6.0, 7.4, 5.9, ↩
 ↪ 6.0, 5.4, 5.2, 4.3, 7.4, 1.2, 1.0]
print(rendement_toets(toetsresultaten), '%')
```

Listing 3: Testcode voor de functie `rendement_toets`, zie `rendement_toets.py`.



Figuur 10: De gewenste uitvoer van de testcode gegeven in listing 3.

B Schrijf een functie genaamd `pas_toetsresultaten_aan` die een rij toetsresultaten steeds met 0.1 ophoogt totdat ten minste een gewenst rendement wordt behaald. Daarbij moet er wel op gelet worden dat het toetsresultaat niet hoger mag worden dan 10.0. De rij met resultaten van de toets wordt als argument, via een Python `list`, doorgegeven aan de functie. Het gewenste rendement wordt als tweede argument aan de functie meegegeven. Maak gebruik van de functie `rendement_toets` die je bij deelopdracht A hebt geschreven. Test deze functie met de testcode uit listing 4. De gewenste uitvoer is gegeven in figuur 11. Waarschijnlijk krijg je echter de uitvoer die gegeven is in figuur 12. Er ontstaan bij het ophogen van de toetsresultaten met 0.1 afrondingsfouten[9]. Deze afrondingsfouten kun je voorkomen door de functie `round` te gebruiken, zie figuur 14.

```
toetsresultaten = [1.2, 3.6, 9.2, 8.2, 3.5, 5.5, 6.0, 7.4, 5.9, ↩
 ↪ 6.0, 5.4, 5.2, 4.3, 7.4, 1.2, 1.0]
print(pas_toetsresultaten_aan(toetsresultaten, 70))
print('Rendement =', rendement_toets(toetsresultaten))
```

Listing 4: Testcode voor de functie `pas_toetsresultaten_aan`, zie `pas_toetsresultaten_aan.py`.

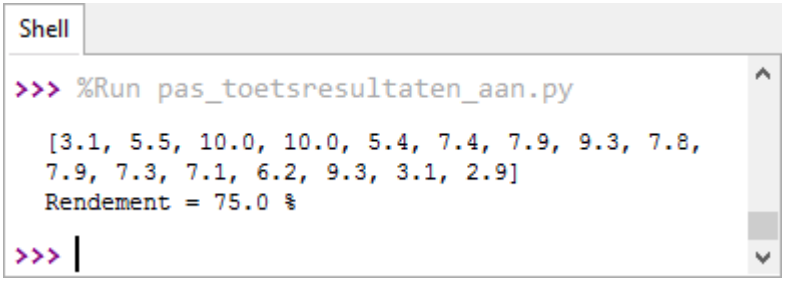
Als je flowcharts wilt maken van de programma's die je bij opdracht 3.1.7 hebt gemaakt, loop je tegen twee problemen aan:

- In Flowgorithm mag je geen liggend streepje gebruiken in een variabele- of functienaam. In Python is het gebruikelijk om bij een naam die uit meerdere woorden bestaat de woorden te scheiden met behulp van een liggend streepje[10]. Deze manier van het samenstellen van woorden wordt ook wel snake_case genoemd[11]. Als alternatief kun

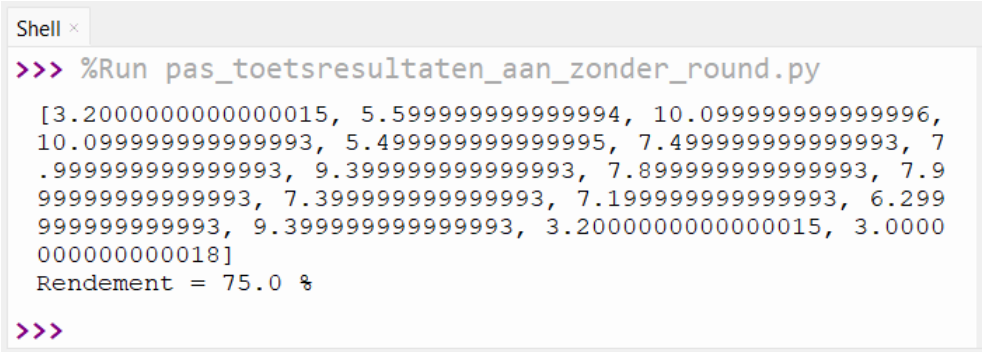
[9] Zie: https://bitbucket.org/HR_ELEKTRO/ems10/wiki/floats.md.

[10] Dit wordt aanbevolen in de *Style Guide for Python*, zie: https://www.python.org/dev/peps/pep-0008/.

[11] Zie eventueel: https://en.wikipedia.org/wiki/Snake_case.



Figuur 11: De gewenste uitvoer van de testcode gegeven in listing 4.



Figuur 12: De vermoedelijke (onjuiste) uitvoer van de testcode gegeven in listing 4.

je in Flowgorithm camelCase gebruiken[12]. De functienaam `rendement_toets` die je in Python hebt gebruikt kun je dus niet gebruiken in Flowgorithm. In plaats daarvan kun je de functienaam `rendementToets` gebruiken.

- In Flowgorithm kun je geen lijsten gebruiken. In plaats daarvan kun je een zogenoemde array gebruiken. De belangrijkste verschillen tussen een lijst in Python en een array in Flowgorithm zijn:
  - Een lijst kan elementen van verschillende typen bevatten. Alle elementen van een array moeten van hetzelfde type zijn.
  - Een lijst is dynamisch, dat betekent dat het programma-elementen aan een lijst kan toevoegen en/of elementen uit een lijst kan verwijderen. Een array is statisch, dat wil zeggen dat het aantal elementen nadat de array is aangemaakt niet meer gewijzigd kan worden.

[12] Zie eventueel: https://en.wikipedia.org/wiki/Camel_case.

  - Een lijst kan bewerkt worden met operatoren en methodes. Een array heeft geen operatoren en geen methodes.
  - De elementen van een lijst kunnen benaderd worden met behulp van indexering met een positieve index (van voor naar achter: index 0, 1, ...) of met een negatieve index (van achter naar voor: index -1, -2, ...). De elementen van een array kunnen alleen benaderd worden met behulp van indexering met een positieve index (van voor naar achter: index 0, 1, ...).
- Python en Flowgorithm kennen beide een **for**-statement. Met behulp van het **for**-statement in Python kun je rechtstreeks door de elementen van de lijst `l` itereren door middel van de code `for e in l:`. In Flowgorithm is dit niet mogelijk. Wel is het in Flowgorithm mogelijk om met behulp van indexering door de elementen van een array die `n` elementen bevat te itereren. Dit komt overeen met de Python code `for i in range(0, n):`.

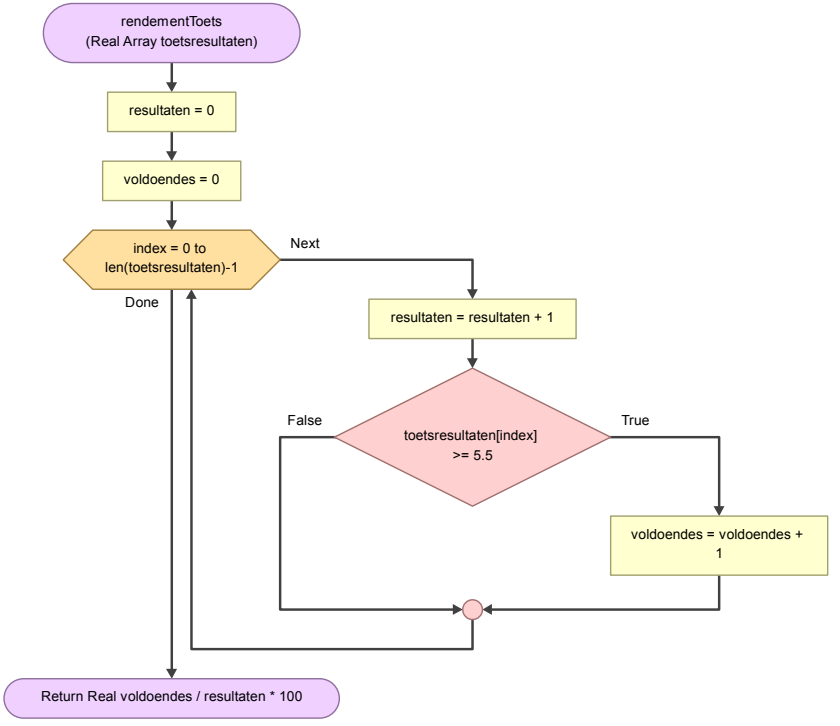
Het programma Flowgorithm sluit beter aan bij de programmeertaal C waarin ook geen lijsten, maar wel arrays standaard beschikbaar zijn. Ook heeft deze programmeertaal een **for**-statement waarmee alleen met behulp van indexering door de elementen van een array geïtereerd kan worden.

3.1.8 Teken flowcharts voor de functies die je bij opdracht 3.1.7 hebt geschreven. Hou daarbij rekening met de bovenstaande beperkingen van Flowgorithm. Mogelijke oplossingen kun je vinden in figuren 13 en 14.

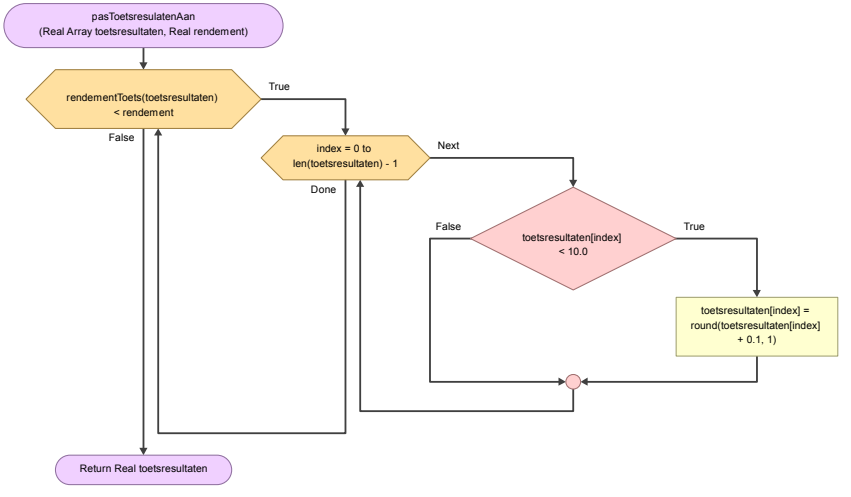
Lees nu paragraaf 10.4, 10.5 en 10.6 van het boek.

3.1.9 Gegeven zijn twee lijsten met woorden, zie listing 5. Schrijf één `print`-statement waarmee het refrein van het bekend kinderliedje "Hoedje van papier"[13] wordt afgedrukt. Maak in het `print`-statement alleen gebruik van de in listing 5 gegeven Python `lists`, de `list`-operatoren `+` en `*` en `list` slices. De gewenste uitvoer is gegeven in figuur 15.

[13] Zie eventueel: https://nl.wikipedia.org/wiki/Hoedje_van_papier.



Figuur 13: Mogelijk flowchart van de functie `rendement_toets`.



Figuur 14: Mogelijk flowchart van de functie `pas_toetsresultaten_aan`.

```
cijfers = ['nul', 'een', 'twee', 'drie', 'vier', 'vijf', 'zes', ↩
 ↪ 'zeven', 'acht', 'negen']
woorden = ['hoedje', 'van', 'papier']
```

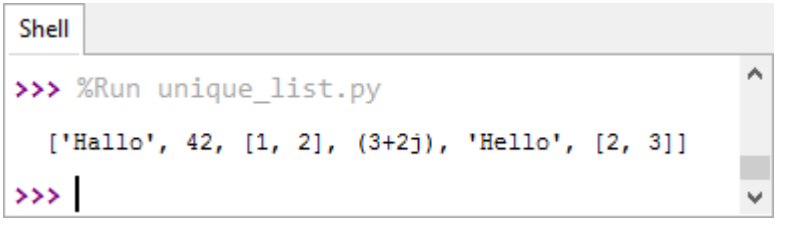
Listing 5: Twee gegeven lijsten met woorden, zie hoedje_van.py.

Figuur 15: De gewenste uitvoer van het te schrijven `print`-statement.

3.1.10 Schrijf een functie genaamd `unique_list` waaraan je een Python `list` als argument kan meegeven en die een nieuwe `list` teruggeeft met daarin alle unieke elementen uit de als argument meegegeven `list`. Test deze functie met de testcode uit listing 6. De gewenste uitvoer is gegeven in figuur 16.

```
test_list = ['Hallo', 42, [1, 2], 42, 3 + 2j, 'Hallo', 'Hello', ↩
 ↪ [1, 2], [2, 3], 3 + 2j, 42]
print(unique_list(test_list))
```

Listing 6: Testcode voor de functie `unique_list`, zie unique_list.py.



Figuur 16: De gewenste uitvoer van de testcode gegeven in listing 6.

Lees nu paragraaf 10.7 van het boek.

3.1.11 In het boek worden operaties op een lijst ingedeeld in map-, filter- en reduceoperaties.

A Voert de functie avg die je bij opdracht 3.1.1 hebt geschreven een map-, filter- of reduceoperatie uit op de als argument meegegeven lijst?

B Voert de functie pas_toetsresultaten_aan die je bij opdracht 3.1.7 hebt geschreven een map-, filter- of reduceoperatie uit op de als argument meegegeven lijst?

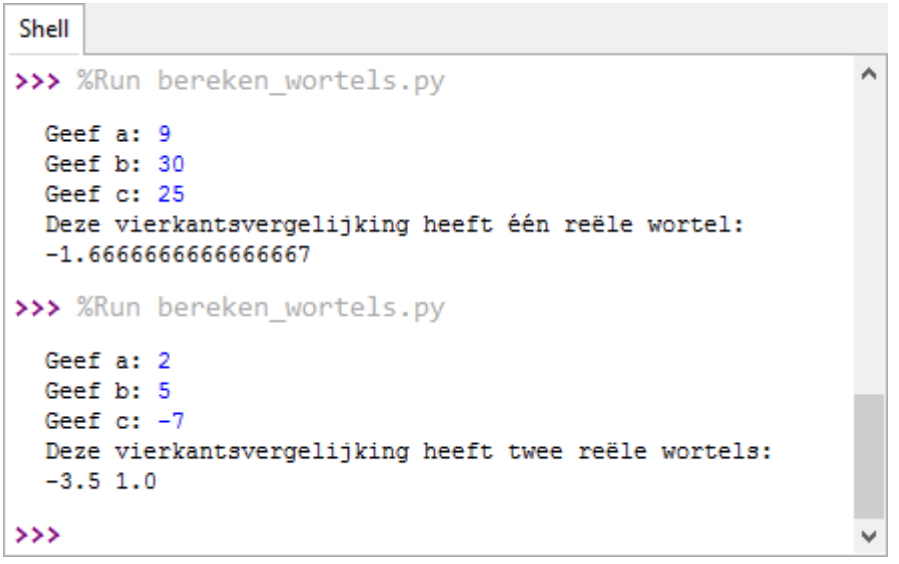
C Voert de functie unique_list die je bij opdracht 3.1.10 hebt geschreven een map-, filter- of reduceoperatie uit op de als argument meegegeven lijst?

3.1.12 Bij opdracht 2.1.5 heb je een functie geschreven die de wortels van de vierkantsvergelijking $a \cdot x^2 + b \cdot x + c = 0$ afdrukt. Vorige week heb je geleerd dat het handiger is als een functie zijn resultaat teruggeeft aan de aanroepende code. Deze code kan het resultaat dan gebruiken in een berekening of indien gewenst afdrukken. Schrijf een functie genaamd bereken_wortels die de wortels van de vierkantsvergelijking $a \cdot x^2 + b \cdot x + c = 0$ berekent en teruggeeft met behulp van een Python list[14]. Deze list kan dus 0, 1 of 2 elementen bevatten. Test deze functie met de testcode uit listing 7. De gewenste uitvoer is gegeven in figuur 17.

```
wortels = bereken_wortels(int(input('Geef a: ')), int(input('Geef ↩
 ↪ b: ')), int(input('Geef c: ')))
aantal_wortels = len(wortels)
if aantal_wortels == 0:
    print('Deze vierkantsvergelijking heeft geen reële wortels.')
elif aantal_wortels == 1:
    print('Deze vierkantsvergelijking heeft één reële wortel:')
    print(wortels[0])
else:
    print('Deze vierkantsvergelijking heeft twee reële wortels:')
    print(wortels[0], wortels[1])
```

Listing 7: Testcode voor de functie bereken_wortels, zie bereken_wortels.py.

[14] Een ervaren Python programmeur zal in dit geval een tuple gebruiken om de wortels terug te geven in plaats van een list. Het datatype tuple is vergelijkbaar maar simpeler dan het datatype list. Een tuple kan na het aanmaken niet meer gewijzigd worden. Het datatype tuple wordt behandeld in hoofdstuk 12 van het boek. Bij EMS10 wordt dit datatype niet behandeld. Als de functie bereken_wortels een tuple in plaats van een list teruggeeft hoeft de testcode die gegeven is in listing 7 niet gewijzigd te worden.



Figuur 17: De gewenste uitvoer van de testcode gegeven in listing 7.

Paragraaf 10.8 tot en met 10.13 van het boek mag je overslaan.

Paragraaf 10.14 van het boek definieert de verschillende (vak)termen die in hoofdstuk 10 gebruikt worden. Het kan handig zijn om deze paragraaf te raadplegen als je niet meer weet wat met een bepaalde (vak)term bedoeld wordt.

Hier eindigen de verplichte opdrachten van week 3 les 1. Er volgen nu nog twee gedeelten:

- oefening, in dit gedeelte vind je extra oefeningen die je helpen om de leerstof van deze les beter te onthouden;
- verdieping, in dit gedeelte vind je uitdagende, verdiepende opdrachten.

Oefening

Veel van de onderstaande oefeningen (met de bijbehorende *uitwerking*) zijn ook beschikbaar op Anki flashcards zodat je ze regelmatig kunt herhalen. Deze stok kun je hier downloaden: EMS10 Week 3 Les 1.apkg.

3.1.13 Geef de uitvoer van het volgende programma[15]:

```
medailles = ['Goud', 'Zilver', 'Brons']
print(medailles[1])
print(medailles[-1])
medailles.sort()
print(medailles)
```

3.1.14 Geef de uitvoer van het volgende programma:

```
van_alles_wat = ['Hallo', 42, 3.1415, [1, 2, 3, 4], 'hoedje ↩
 ↪ van papier']
print(van_alles_wat[1] + van_alles_wat[3][-1])
print(len(van_alles_wat))
print(len(van_alles_wat[3]))
print(van_alles_wat[:2])
print(van_alles_wat[3][1:3])
print(van_alles_wat[-1][7:10])
```

3.1.15 Geef de uitvoer van het volgende programma:

```
getallen = [0, 2, 3, 5, 0]
getallen[0] = getallen[2] - getallen[1]
getallen[-1] = getallen[-2] + getallen[1]
print(getallen)
print(sum(getallen))
print(sum(getallen[1:-1]))
getallen.append(11)
print(getallen)
getallen.extend([13, 17])
print(getallen)
```

3.1.16 Geef de Pythoncode om de getallen uit de list `lijst` één per regel af te drukken.

3.1.17 Geef de Pythoncode om het aantal getallen in de list `lijst` af te drukken.

3.1.18 Geef de Pythoncode om alle getallen in de list `lijst` met één te verlagen.

[15] Het is de bedoeling om de uitvoer te bepalen door het programma 'in je hoofd' uit te voeren. Je kunt het programma *daarna* uitvoeren met Thonny om je antwoord te controleren.

3.1.19
```
van_alles_wat = ['Hallo', 42, 3.1415, [1, 2, 3, 4], 'hoedje van papier']
print(van_alles_wat[1] + van_alles_wat[3][-1])
print(len(van_alles_wat))
print(len(van_alles_wat[3]))
print(van_alles_wat[:2])
print(van_alles_wat[3][1:3])
print(van_alles_wat[-1][7:10])
```

3.1.20 Geef de uitvoer van het volgende programma:

```
print(([1, 2] + [3] * 2) * 3)
```

3.1.21 Geef de uitvoer van het volgende programma:

```
getallen = [0, 2, 3, 5, 0]
getallen[0] = getallen[2] - getallen[1]
getallen[-1] = getallen[-2] + getallen[1]
print(getallen)
print(sum(getallen))
print(sum(getallen[1:-1]))
getallen.append(11)
print(getallen)
getallen.extend([13, 17])
print(getallen)
```

3.1.22 Geef de uitvoer van het volgende programma:

```
def wtf(lijst):
    res = []
    for element in lijst:
        res = [element] + res
    return res

print(wtf([1, 2, 3]))
```

Als je nog meer wilt oefenen, dan kun je gebruik maken van sololearn[16]. Deze app kun je ook op je telefoon installeren. De cursus “Python for Beginners” bevat veel eenvoudige oefenstof.

[16] https://www.sololearn.com

Als je een eigen account aanmaakt, dan moet je alle oefeningen in de aangegeven volgorde doen (of betalen). Je kunt beter gebruik maken van het account `eleems10@gmail.com` met het wachtwoord `ELEEMS10`, dan kun je meteen naar het onderdeel dat je wilt oefenen.

Paragraaf 10.15 van het boek bevat vele oefeningen. Oefeningen 10.1 tot en met 10.7 zou je op dit moment moeten kunnen maken. Je kunt ze gebruiken als extra oefenmateriaal.

Verdieping

Paragraaf 10.15 van het boek bevat vele oefeningen. Oefeningen 10.8 tot en met 10.12 zijn pittiger. Studenten die een extra uitdaging zoeken kunnen zich uitleven met deze opdrachten, maar het is niet noodzakelijk om ze te maken.