FACULTEIT WETENSCHAPPEN

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

NASCHOLING ALGORITMEN EN PROGRAMMEREN

Module 1 Basiskennis programmeren in Python

Oefeningenbundel

Leen Brouns

UNIVERSITEIT GENT

IPVW ACADEMIE VOOR LEVENSLANG LEREN

Inhoudstafel

1 Programmeeromgeving Thonny 4
2 De shell als experimenteerruimte 4
3 De shell als hulpbron 7
4 Een eerste programma 7
5 Interactie met de gebruiker 8
6 Tekst bewerken 9
7 Spiekbriefje 9
8 Getallen inlezen 9
9 Namen van variabelen 9
10 Type van waarden van variabele 10
11 Toekenning van waarden aan variabelen 11
12 Vierkantswortel indien niet negatief 12
13 Begroeting 12
14 Begroeting met extra keuzes 13
15 Keuzestructuur met dubbele voorwaarden 13
16 Keuzestructuur met een leeg deel 13
17 Blad, steen, schaar 14
18 Dobbelspel 14
19 Een variabele updaten 15
20 Gevallenstudie 16
21 Van if naar while 16
22 Input afgesloten door stopwaarde 16
23 Plaats van code ín de lus 17
24 Controle op input van gebruiker 17
25 Keuzestructuur voorafgegaan door controle op input 18
26 Debugtips 18
27 Herhaling waarbij aantal gekend is 19
28 Herhaling waarbij teller gebruikt wordt 19
29 Sommeren 19
30 Wanneer welke lus? 20
31 De opgave bepaalt de soort lus 20
32 Dubbele voorwaarden 20
33 Lussen combineren 21

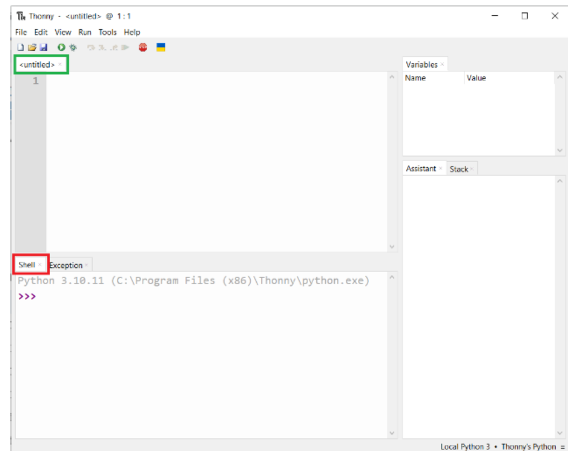
34 While-lus opkuisen 21
35 Strings verkennen 23
36 Palindroom 24
37 Strings overlopen 24
38 Slicing 24
39 Strings overlopen en letters verwerken 25
40 Huiswerk tegen Lesdag 2 25
FUNCTIES 26
101 Gebruik van functies herkennen 26
102 Functies definiëren en schetsen 26
103 Functies schrijven: Romeinse cijfers 27
104 Functies in functies: volkomen kwadraten 28
105 Functies in functies: van specifiek naar algemeen 28
106 Het nut van functies 29
TEKST 30
107 Methodes op tekst 30
108 Tekst opsplitsen en weer samenvoegen 30
109 Methodes op tekst en slicing 30
110 Alliteraties 30
LIJSTEN 32
111 Lijsten overlopen en uitschrijven 32
112 Lijsten overlopen en elementen verwerken 32
113 Elementen in nieuwe lijst steken 32
114 Mediaan 32
115 Stringmethodes versus Listmethodes 33
116 Frequenties 34
Uitbreiding 34
117 Totems 34
118 Desserts tellen 35
119 Elementen tellen die aan voorwaarde voldoen 35
120 Lijsten: foreach-lus versus for-lus 35
121 Lijsten: gelijktijdig doorlopen 36
122 Lijsten: berekening op basis van voorgaand of volgend element 36
123 Desserts uitdelen aan juiste persoon 37
124 Tekst vervangen 38
125 Wanneer zijn lijsten nodig / nuttig? 38

126 Lijsten mixen 39
Uitbreiding: testdata uit bestand 40
127 Structuur van de while-lus 40
128 De speld in de hooiberg 41
Uitbreiding: hooibergdata uit bestand 41
129 Interpolatie 42
130 Rijmwoorden 42
Voorbereiding 42
Rijmwoorden 43
131 Recursie 44
132 Kritische zin aangescherpt 44
Oplossingen 44

Basiskennis programmeren in Python
Lesdag 1

1 Programmeeromgeving Thonny

Surf naar https://thonny.org/ en download Thonny, een Python IDE voor beginners. De afkorting IDE staat voor *integrated development environment*, en betekent dat Thonny een stuk software is dat een programmeur, die nieuwe programma's (of nieuwe software) ontwikkelt, een omgeving aanbiedt waarin dit makkelijk kan. Er zijn ook verschillende hulpmiddelen geïntegreerd in Thonny, die onderdelen zullen we stelselmatig tegenkomen.



Bij het openen van Thonny zijn er een aantal vensters te zien. We bespreken eerst de twee linkervensters.

Het venster linksonder toont momenteel de shell (zie rode kader in de afbeelding hierboven). Daar kan je telkens een regel Python-code intikken, die dan onmiddellijk gelezen ('geïnterpreteerd') en uitgevoerd wordt. Vergelijk het met de werking van een rekenmachine: na het ingeven van het gelijkheidsteken, krijg je onmiddellijk het antwoord te zien op het scherm.

Het venster linksboven draagt momenteel nog de naam <untitled> (zie groene kader in de afbeelding hierboven). In dit venster, de editor, schrijven we later Python-code die een volledig afgewerkt programma vormt. We zullen de inhoud van dit venster dan ook bewaren in een bestand (zie later), zodat het later teruggevonden kan worden. Dat is het verschil met de shell: wat in de shell gebeurt, is tijdelijk van aard; wat in de editor staat, wordt bewaard voor later.

2 De shell als experimenteerruimte

De shell werkt als *command line interface (CLI)*. Dat is een editeeromgeving die op tekst gebaseerd is. Die tekst, indien welgevormd, vormt commando's waarop de computer zal reageren.

Er staan al drie pijltjes >>> klaar. Die noemen we de *prompt*. Die prompt spoort de programmeur aan om Python-code (of een andere vraag) in te tikken, en af te sluiten met enter. Dan zal die code onmiddellijk uitgevoerd worden. In deze lesnota's zal de prompt ook telkens afgeprint worden als >>>; uiteraard moet je die pijltjes niet overtikken want die staan al klaar in de shell.

Tik in de shell de bewerking 2 + 2.5 en klik op enter. Er komt:

```
>>> 2 + 2.5
4.5
```

Je kunt ook met variabelen werken. Een variabele is een (leesbare, betekenisvolle) naam die je aan een geheugenplaats geeft. In die geheugenplaats kan je dan een waarde (bijv. een getal) bewaren, zodat dat getal later nog geraadpleegd kan worden. Omdat de eerste oefeningen voornamelijk om eenvoudige wiskundige bewerkingen gaan, zijn de namen van de variabelen hier heel kort. Later wordt dat anders. Tik in wat achter de drie pijltjes staat.

```
>>> x=20/2
```

Nu wordt er een geheugenplaats gereserveerd, die krijgt de naam x, en de waarde 20/2 (of 10) wordt in die geheugenplaats bewaard. Let op: de variabelenaam moet in het linkerlid staan. Lees het gelijkheidsteken (=) als *'krijgt de waarde van'*. Je mag spaties tussenvoegen, graag zelfs: het bevordert de leesbaarheid.

```
>>> x = 20 / 2
```

Let ook op kleine letters en hoofdletters; x is niet hetzelfde als X.

Merk op dat de shell niet echt reageert: nadat je op enter tikt, komt er een nieuwe prompt vlak na de ingetikte regel. Het getal `10` is niet te zien. Er staat dus

```
>>> x = 20 / 2
>>>
```

De gegeven opdracht werd in de achtergrond uitgevoerd, zonder dat er effect te zien is. De prompt zal wél een reactie geven als je één van deze drie zaken intikt:

- een bewerking op getallen (bijv. 7 + 2)
- de naam van een variabele (bijv. x, als x al gekend is als variabele)
- een bewerking op variabelen (bijv. 10 * a + b, als a en b al gekend zijn als variabelen)

Probeer dat uit, vraag de waarde van x op:

```
>>> x
10.0
```

Merk op: er wordt .0 uitschreven om aan te geven dat de uitkomst een reëel getal is (een kommagetal). Dat wordt later nog duidelijk.

Je kunt x nu ook gebruiken in bewerkingen:

```
>>> x + 3
13.0
```

Vermenigvuldiging en deling hebben voorrang op optellen en aftrekken; gebruik haakjes om de volgorde te wijzigen:

```
>>> 2 + 3 * 5
17
>>> (2 + 3) * 5
25
```

Wat is de waarde van `2 ** 3` ? Wat is dan de betekenis van de operator ** ? Controleer het antwoord met andere waarden voor linker- en rechteroperand (de linkeroperand is de uitdrukking die links van de operator staat; de rechteroperand staat er rechts van).

We kunnen ook wiskundige functies gebruiken. Die zitten echter niet standaard in de programmeertaal Python, maar zijn verpakt in een specifieke module. De module die de wiskundige functies groepeert, draagt de naam *math* (van *mathematics*). In die module wordt er gedefinieerd wat er moet gebeuren als er bepaalde functies (zoals `cos`, `sin`, `exp`, `log`,...) opgeroepen worden. Als gebruiker moet je niet weten wat er achter de schermen gebeurt, je moet alleen weten hoe je die functies kan gebruiken.

Wiskundige functies gebruiken, gebeurt in twee stappen: eerst wordt er aangegeven dat de module math gebruikt zal worden. Dat kan met de opdracht import:

```
>>> import math
```

Deze opdracht dient niet herhaald te worden; het is voldoende dat die één keer gegeven wordt.

Nadien verloopt het oproepen van een functie zoals in de wiskunde-les: de cosinus van 0 wordt berekend als `math.cos(0)`. Merk op: schrijf de naam van de module nog wel voor de naam van de functie.

```
>>> math.cos(0)
1.0
```

Het argument (hier `0`) moet tussen haakjes staan. De schrijfwijze uit de wiskundeles (`cos 0`) is niet geldig, en levert een syntaxfout op.

Ook bepaalde wiskundige constanten zijn terug te vinden in de module math:

```
>>> math.sin(math.pi/2)
1.0
```

Heb je in het hele programma enkel de cosinus nodig, dan kan je ook enkel die functie importeren. (Te onthouden voor later, als we de shell inruilen voor de editor: schrijf import-opdrachten altijd helemaal bovenaan het programma.)

```
>>> from math import cos
>>> cos(0.2)
0.9800665778412416
```

In Python wordt de delingsoperator / gebruikt voor de reële deling. Dus de input 7/2 levert output 3.5.

```
>>> 7 / 2
3.5
```

Als je in Python het afgekapte resultaat wil, moet je de operator gebruiken die in Python een gehele deling aanduidt: //.

```
>>> 7 // 2
3
```

Wil je niet het *quotiënt* (3) maar de *rest* bij deling door 2, dan gebruik je de modulo-operator %.

```
>>> 7 % 2
1
```

Voorspel wat onderstaande bewerkingen zullen opleveren en controleer.

```
>>> 789 % 100
>>> 123 // 100
>>> 6789 // 100
>>> 35 % 5
>>> 7 % 12
>>> 100 // 123
```

3 De shell als hulpbron

In de shell kan ook om hulp gevraagd worden.

```
>>> help()
```

Lees aandachtig wat er als antwoord komt.

Verlaat daarna de help-modus van de shell met de opdracht quit.

```
>>> quit
```

Daarna kan je specifieker zijn in de hulpvraag: vraag informatie over de module math.

```
>>> help(math)
```

Indien er een foutmelding komt, dan is de module math allicht nog niet geïmporteerd. Dan volstaat het om eerst `import math` te schrijven, en daarna opnieuw te proberen met `help(math)`.

Scrol door het lijstje van functies in de math-module. Bekijk de functie `cos`. De parameterlijst (de opsomming die tussen de ronde haakjes staat na de naam van de functie), geeft aan dat er één parameter is, net zoals in de wiskundeles.

Zoek daarna op wat de functie `sqrt` doet. Hoeveel parameters neemt deze functie?

Zoek tot slot de functie `comb`. Hoeveel parameters neemt deze functie? (Meer moet je niet verstaan van deze functie; die zal niet gebruikt worden in het vervolg van de oefeningen.)

Scrol verder naar beneden, tot je de prompt weer ziet staan. Bij het opvragen van documentatie met de opdracht `help(math)` krijg je alle details. Ben je alleen geïnteresseerd in een opsomming van alle constanten en functies, dan kan je deze opdracht gebruiken:

```
>>> dir(math)
```

4 Een eerste programma

Korte stukjes code, om uit te testen hoe ze werken, worden ingetikt in de shell. Om echte programma's te schrijven die herhaaldelijk uitgevoerd kunnen worden (zonder opnieuw alle Python-opdrachten in te tikken), moet de geschreven code opgeslagen worden.

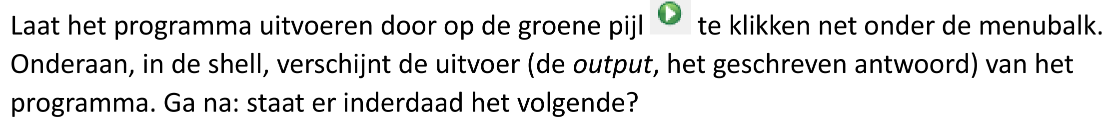
Om zeker te zijn dat deze code nadien nog gelokaliseerd kan worden, is het geen slecht idee om eerst, buiten Thonny, in een bestandsverkenner een map aan te maken voor de komende programma's. Open een bestandsverkenner, maak een map aan voor deze nascholing, en daarbinnen eventueel een submap voor de eerste nascholingsmodule.

Keer terug naar Thonny, en bewaar het (lege) bestand dat nu te zien is in het venster linksboven. Open daarvoor in het menu *File > Save*, en bewaar het (lege) programma onder een passende naam (bijv. *oef01*, als je de oefeningen graag op nummer houdt). Merk op dat de bestandsextensie automatisch .py (van Python) wordt.

Nu zijn we klaar om in dit venster, de editor, code te schrijven. Zodadelijk kan opgemerkt worden dat de editor automatisch kleur zal gebruiken voor bepaalde onderdelen van de code. Zo kunnen gereserveerde sleutelwoorden, constanten, letterlijke tekst,... snel onderscheiden worden van elkaar. Ook bij het juist plaatsen van de kantlijnen (het indenteren van de code) biedt de editor ondersteuning.

Het eerste programma bestaat uit een toekenning (we kennen de variabele x de waarde 4 toe), en twee print-opdrachten.

```
x = 4
print("hallo!")
print(x + 1/x)
```

Laat het programma uitvoeren door op de groene pijl  te klikken net onder de menubalk. Onderaan, in de shell, verschijnt de uitvoer (de *output*, het geschreven antwoord) van het programma. Ga na: staat er inderdaad het volgende?

```
hallo!
4.25
```

Merk op dat de dubbele aanhalingstekens rond de tekst `hallo!` niet uitgeprint worden. Die aanhalingstekens (of *quotes*) zijn wel nodig in de Python-code. Deze quotes geven aan dat alles wat er tussen staat, moet geïnterpreteerd worden als letterlijke tekst. De opdracht `print("hallo!")` schrijft dus de letterlijke tekst die tussen de quotes staat, uit in de shell (het outputvenster).
De tweede printopdracht schrijft geen letterlijke tekst uit – want er zijn geen aanhalingstekens te bespeuren in de opdracht `print(x + 1/x)`. De naam `x` verwijst hier naar de variabele `x`. Eerst wordt de som `x + 1/x` berekend, waarna het resultaat afgeprint wordt.

5 Interactie met de gebruiker

Een programma is een stappenplan dat gevolgd dient te worden om, op basis van een bepaalde input, een bepaalde output te berekenen.

Als voorbeeld uit het dagelijks leven nemen we volgend weekendprogramma van de familie Klepkens:

> *We gaan naar zee als de zon schijnt, en we lezen een boekje als het regent*.

We weten pas effectief tot welke output dit programma geleid heeft, als het weekend aangebroken is en het weerbericht bekeken is. Maar ook elk komend weekend kan ditzelfde programma uitgevoerd worden, met af en toe een andere uitkomst – gebaseerd op de input van het weerbericht.

Het eerste Python-programma dat hierboven geschreven werd, had geen echte input (= informatie die bij elke doorloop van het programma anders kan zijn). Elke keer als het programma

```
x = 4
print("hallo!")
print(x + 1/x)
```

uitgevoerd (of gerund) wordt, zal de output

```
hallo!
4.25
```

zijn.

Laten we hier verandering in brengen. Het programma vraagt aan de gebruiker om diens naam op te geven, en schrijft een gepaste begroeting uit.

```
naam = input("Hallo, met wie spreek ik? ")
begroeting = "Dag " + naam + "!"
```

```
print(begroeting)
```

Neem dit programma over (de vorige code mag weg), en voer het uit. Het programma blijft nu eventjes hangen, tot de gebruiker (= jij) een naam ingetikt heeft, en op enter heeft geduwd. Hieronder staat de hele output van het programma; wat de gebruiker intikte staat vetgedrukt.

```
Hallo, met wie spreek ik? Leen
Dag Leen!
```

Als Hanne Noor het programma uitvoert, dan komt er

```
Hallo, met wie spreek ik? Hanne Noor
Dag Hanne Noor!
```

6 Tekst bewerken

In bovenstaande oefening gebruikten we een variabele om een stuk tekst in te bewaren. Voorspel wat volgende code zou kunnen doen.

```
aanhef = "hip"
print( 3 * aanhef + "hoera")
```

Tik in en controleer. Het resultaat is nu niet zo leesbaar, want alle woorden plakken aan elkaar. Verander het programma op één plaats (voeg één karakterteken toe), en zorg er zo voor dat de output uit vier losse woorden bestaat.

7 Spiekbriefje

Net zoals een formuleblad in de wiskunde- of fysicales een hulpmiddel kan zijn bij het maken van oefeningen, is een overzicht van de meest gebruikte Python-code aan te bevelen in de informaticales. Neem de eerste drie delen van het spiekbriefje door: *'help'*, *'input en output'* en *'toekenning en wiskundige operatoren'*. Fluoresceer of vink af wat al aan bod kwam in de vorige paragrafen. Doe dit ook stelselmatig bij de volgende oefeningen.

8 Getallen inlezen

Schrijf een programma dat de gebruiker twee gehele getallen vraagt. Er moet niet gecontroleerd worden of de gebruiker doet wat gevraagd wordt; er mag vanuit gegaan worden dat de gebruiker het spel goed meespeelt.

Schrijf dan de som van de twee getallen uit.

De output van het programma zou er als volgt kunnen uit zien (input van de gebruiker in het vet):

```
Geef eerste getal: -5
Geef tweede getal: 15
10
```

9 Namen van variabelen

Een variabele is een leesbare, zinvolle benaming van een geheugenplaats. De programmeur kiest zelf de namen van de variabelen die nodig zijn in het programma, en waakt dus zelf over de zinvolheid van de variabelenaam. De naam moet echter ook aan een aantal regels voldoen; hij moet *welgevormd* zijn volgens de voorschriften van de gebruikte programmeertaal. Een aantal richtlijnen (iets strikter dan wat online gevonden kan worden):

- de naam van een variabele bevat enkel letters, cijfers en underscores (= liggende streepjes).
- de naam van een variabele start met een letter, nooit met een cijfer. (De ontwikkelaars van Python mogen ook een underscore gebruiken als eerste karakterteken.)
- de naam van een variabele wordt altijd aan elkaar geschreven; samengestelde woorden bevatten een underscore (bijv. omtrek_cirkel).
- namen zijn hoofdlettergevoelig: leeftijd, Leeftijd, leefTIJD en LEEFTIJD zijn vier verschillende variabelen. Voor de goede verstandhouding: gebruik enkel kleine letters, dan is er geen verwarring mogelijk.
- gebruik geen sleutelwoord van de programmeertaal waarin je bezig bent, die woorden zijn gereserveerd. Een opsomming staat online (volg de link of zoek met de zoektermen *Python keywords*).
- hergebruik nooit de naam van een gekende of zelfgeschreven functie als variabele; dat is vragen om problemen.

Hieronder staan een aantal namen van variabelen. Hier werd gekozen om alle variabelen een Nederlandse naam te geven. Dat is een keuze die voor beginners aangewezen is. Zo is het veel duidelijker wat al zeker de naam van een variabele (of een zelfgemaakte functie) is (Nederlands) en wat een term is uit de programmeertaal (deze gereserveerde Python-termen zijn Engels).

Duid met een kruis aan welke namen ongeldig zijn, en die dus syntaxfouten zullen opleveren.
Duid met een sad smiley aan welke namen beter vermeden worden.

```
getal
negatief getal
3voud
d
kleur
Schoenmaat
GROOTTE
_lengte
al@rm
gokje!
```

10 Type van waarden van variabele

De mogelijke waarden die aan een variabele worden toegekend, kunnen onderverdeeld worden in vier verschillende types.

- Een stuk tekst, bvb `"hallo"`, is van type **`str`** (voor *string*, Engels voor aaneenschakeling (van letters))
- Een geheel getal, bvb `-5`, is van type **`int`** (voor *integer*, Engels voor geheel getal)
- Een reëel getal, bvb `7.77`, is van type **`float`** (vertaald: vlottende-kommagetal)
- De waarden `True` en `False` noemen we logische waarden of waarheidswaarden, die zijn van type **`bool`** (van booleaans, zoals in Booleaanse algebra of logica)

In Python kan elke variabele een waarde aannemen van eender welk type. Het is het *rechter*deel van de toekenning

```
a = "hallo"
```

die vastlegt dat, in dit geval, de variabele a een stuk tekst bevat. Maar evengoed kan de inhoud van de variabele in de volgende regel vervangen worden door een getal:

```
a = 17
```

Het is wel raadzaam om de namen van de variabelen zinvoller te kiezen, zodat het 'hergebruik' van een variabele (nu eens voor tekst, dan weer voor een getal) vermeden wordt. Het vraagt anders teveel speurwerk in de code om te weten van welk type de meest recent toegekende waarde is. Desnoods wordt een extra variabele ingeschakeld:

```
begroeting = "hallo"
leeftijd = 17
```

In een van de vorige oefeningen werd duidelijk dat de waarde van de rechteroperand in de toekenning

```
getal = input("Geef een getal")
```

van type `str` is: een stuk tekst. Maar dat is niet het type dat we verwachten in een variabele met de naam `getal`. Daarom wordt de rechteroperand eerst nog omgezet: de *string* wordt gecast (= omgezet) naar een *geheel getal*, met behulp van de functie `int(...)`:

```
getal = int(input("Geef een getal"))
```

In alle volgende oefeningen zullen we er vanuit gaan dat de gebruiker ook doet wat gevraagd wordt: hij geeft een getal, als er een getal gevraagd wordt.

Stel nu dat volgend programma gegeven is:

```
a = int(input("Geef invoer: "))
print(a/2)
```

Omcirkel voor welke invoer, ingetikt door de gebruiker, het programma fout loopt:

a) True
b) 5
c) 7.8
d) prinskoek

Opmerking voor gevorderden: een gebruiker kan nooit de woorden True of False intikken, en hopen dat er dan eenvoudig een automatische omzetting kan gebeuren naar het type `bool`, zoals dat kan met het casten naar een `int` of een `float` (waarbij een woord (type `str`) wordt omgezet naar een geheel getal (type `int`) of een reëel getal (type `float`)). Dus niet proberen.

11 Toekenning van waarden aan variabelen

Zet twee variabelen klaar, en bewaar in elk van die variabelen een woord. Dat mag *hardgecodeerd*, dat wil zeggen dat de programmeur van de code beslist welke inhoud de variabelen krijgen. Het programma vraagt dus niet aan de gebruiker om de woorden in te vullen, maar beslist zelf, bijvoorbeeld:

```
woord_1 = "appelbol"
woord_2 = "kerst"
```

Wissel daarna de inhoud van de variabelen. Schrijf daarna de variabelen uit, om te zien of de inhoud inderdaad gewisseld werd. (Uiteraard wordt er niet vals gespeeld, niet de output van het programma maar de code van het programma levert punten op!)

Een voorbeeld van de output als het programma doet wat gevraagd werd:

```
Eerst komt woord_1, dan komt woord_2:
appelbol
kerst
Eerst komt woord_1, dan komt woord_2:
kerst
appelbol
```

Voeg tot slot ook commentaar toe aan het programma, zodat het leesbaar is voor anderen (of voor jezelf, binnen een paar weken). Commentaar wordt voorafgegaan door een hashtag. Het is alvast een goed idee om volgende onderdelen aan te duiden:

```
# invoer
...
# uitvoer voor het wisselen
...
# wisselen
...
# uitvoer na het wisselen
...
```

Daarnaast kan er ook wat commentaar komen bij de onderdelen die moeilijk waren om te schrijven: als er een moeilijk 'algoritme' (of stappenplan) uitgedacht werd, verdient het een woordje uitleg.

Keuzestructuur

Lees in het spiekbriefje de drie onderdelen *'vergelijkingsoperatoren op int, float, str'*, *'logische operatoren'* en *'keuze: if-structuur'.*

12 Vierkantswortel indien niet negatief

Lees een getal in, en bereken de vierkantswortel van dat getal als het positief is. Als het negatief is, komt er een passende boodschap.

Voorbeeld van het programmaverloop:

```
Geef een getal: 10
3.1622776601683795
```

Ander voorbeeld van het programmaverloop:

```
Geef een getal: -10
Dit getal is negatief.
```

13 Begroeting

Vraag aan de gebruiker of hij M, V of X is. Ga er vanuit dat de gebruiker één van deze drie ingeeft (met hoofdletters). Daarna begroet je de gebruiker met de gepaste aanspreking:

- Geachte heer (voor M)
- Geachte mevrouw (voor V)
- Geachte lezer (voor X)

Voorbeeld van het programmaverloop:

```
Kies uit M, V of X: X
Geachte lezer,...
```

14 Begroeting met extra keuzes

Herhaal dezelfde oefening, maar laat toe dat de gebruiker ook met kleine letters antwoordt. Dus M, V, X, m, v en x zijn de mogelijkheden. Als de gebruiker toch nog iets anders kiest, dan komt er een passende boodschap.

Voorbeeld van het programmaverloop:

```
Kies uit M, m, V, v, X of x: v
Geachte mevrouw,...
```

Ander voorbeeld van het programmaverloop:

```
Kies uit M, m, V, v, X of x: y
Mijn excuses, ik ben even de kluts kwijt.
```

Vergelijk nu jouw oplossing met die van de buur. Welke oplossingen zijn leesbaar? Waar zit het verschil?

15 Keuzestructuur met dubbele voorwaarden

Gegeven volgende code, die bepaalt of er een schrikkeljaar werd opgegeven of niet. Er zijn maar twee verschillende uitkomsten mogelijk. Herschrijf de code zo, dat er slechts één `if`- en één `else`-gedeelte is. (Om te weten wat de precieze regels zijn voor schrikkeljaren, lees je dit programma óf raadpleeg je Wikipedia.)

```
jaar = int(input("Geef een jaartal op. "))
if jaar % 4 != 0 :
    print("geen schrikkeljaar")
else:
    if jaar % 100 == 0:
        if jaar % 400 == 0:
            print("schrikkeljaar")
        else:
            print("geen schrikkeljaar")
    else:
        print("schrikkeljaar")
```

Tip: maak een diagram waarin de verschillende situaties visueel voorgesteld worden. Vertrek daarvoor van de gegeven code.

16 Keuzestructuur met een leeg deel

Hieronder staat een onafgewerkt programma dat werd geschreven naar aanleiding van deze opdracht:

> *Schrijf een programma dat een woord inleest. Als het woord minder dan tien letters heeft, dan gebeurt er niets. Anders wordt het woord vervangen door de tekst `"te lang!"`. Het (al dan niet gewijzigde) woord wordt uitgeschreven.*

```
woord = input("Geef een woord: ")
if len(woord) < 10:     # functie len berekent lengte van het woord
    # doe niets, hoe schrijf je dat ??
else:
    woord = "te lang!"
print(woord)
```

Herschrijf het programma: zorg dat de vraagtekens weg zijn.

17 Blad, steen, schaar

Schrijf een programma dat het spel *blad, steen, schaar* speelt. Het programma vraagt de gebruiker wat de eerste speler gekozen heeft (blad, steen of schaar). Idem voor de tweede speler. Daarna schrijft het programma uit welke speler gewonnen is (of misschien was het gelijkspel...). Nog even de regels: de schaar verknipt het blad, het blad omwikkelt de steen, de steen maakt de schaar bot.

Een mogelijk programmaverloop:

```
Wat koos speler 1 (blad,steen,schaar)? blad
Wat koos speler 2 (blad,steen,schaar)? blad
het is gelijkspel
```

Een ander programmaverloop:

```
Wat koos speler 1 (blad,steen,schaar)? blad
Wat koos speler 2 (blad,steen,schaar)? schaar
speler 2 wint
```

18 Dobbelspel[1]

Schrijf een programma dat een dobbelspel met drie dobbelspelen speelt. Start het programma met volgende code:

```
import random

a = random.randint(1,6)
b = random.randint(1,6)
c = random.randint(1,6)

print(a)
print(b)
print(c)
```

Een beetje uitleg hierbij: de module random bevat een aantal functies die te maken hebben met het genereren van random (= willekeurige) getallen. De functieaanroep

```
random.randint(onder,boven)
```

zal een geheel getal teruggeven dat in het gesloten interval `[onder, boven]` ligt. De code die hier staat, zorgt dus voor drie variabelen die elk een willekeurig getal uit de verzameling {1,2,3,4,5,6} bevatten. Elke variabele stelt dus een worp met één dobbelsteen voor.

---

[1] Deze oefening is er eenje die zich leent tot 'gespreid oplossen': de opgave lanceren, er een nacht over laten gaan, dan pas coderen.

Bij een dobbelspel is het dikwijls de bedoeling om zoveel mogelijk gelijke waarden te gooien. Of net opeenvolgende waarden, wat dan 'drie op een rij' genoemd wordt.

Vervolledig het programma: zorg ervoor dat één van de volgende boodschappen uitgeschreven wordt, naargelang de situatie die zich voordoet:

```
alle drie gelijk
twee gelijke
drie op een rij
geen speciaal trio
```

Een tip: welk verband is er tussen het kleinste en het grootste getal bij *'drie op een rij'*?

Om het programma te testen zijn er twee opties:

- ofwel voer je het verschillende keren uit, en kijk je nauwkeurig na of elke situatie goed herkend wordt.
- ofwel geef je a, b en c voorlopig hardgecodeerde waarden, om na te gaan of een specifiek geval goed herkend wordt. Die methode van testen is het zekerst. Zorg er wel voor dat het lijstje van alle mogelijke gevallen zorgvuldig werd opgesteld en afgelopen.

19 Een variabele updaten

Stel dat in de wiskundeles de formule

```
x = x * 10
```

op bord staat. Wat besluit je daar dan uit? Allicht komt er dan

```
⇒  x = 0
```

En wat als de formule

```
a = a + 5
```

op bord staat? In de informaticales wordt dit helemaal anders gelezen. Om te beginnen staat hier geen formule maar een toekenningsopdracht. Het teken '=' is geen gelijkheidsteken, maar een toekenningsteken. Dus wordt

```
a = a + 5
```

gelezen als

de variabele a krijgt de waarde van zichzelf, vermeerderd met 5

De toekenning kan ook gelezen worden van rechts naar links: eerst wordt de bewerking `a + 5` uitgevoerd, waarna dat resultaat toegekend wordt aan de variabele a. De *vorige* waarde van a wordt dus overschreven door het resultaat, de *volgende* waarde van a.

Gebruik dit in volgende opdracht, waar ergens een '*volgende*' waarde uit een '*vorige*' waarde berekend dient te worden.

Schrijf een programma dat een woord inleest. Als het woord korter is dan tien letters, dan worden er drie uitroeptekens achter geplakt. Het (al dan niet gewijzigde) woord wordt uitgeschreven. (De lengte van een woord werd al eens gebruikt in oefening 16.)

20 Gevallenstudie

Gegeven de opgave:

> *Schrijf een programma dat een geheel getal inleest. Als het kleiner is dan 10, dan wordt er 10 bij opgeteld. Als het tussen 10 en 20 ligt (grenzen inbegrepen), dan wordt het verdubbeld. Als het groter is dan 20, dan wordt er 1000 bij opgeteld.*

Gegeven dit voorstel voor oplossing:

```
x = int(input("Geef een getal: "))
if x < 10:
    x = x + 10
if 10 <= x <= 20:
    x = 2 * x
if 20 < x:
    x = 1000 * x
print(x)
```

Wat gaat er fout en waarom? Som op welke input je uitprobeert, dus houd het lijstje testgevallen goed bij. Verbeter de code.

Herhalingsstructuur

21 Van if naar while

Gegeven onderstaande code.

```
getal = int(input("Geef een getal: "))
if getal < 100:
    getal = getal + 10
print(getal)
```

Neem deze code over, en test uit. Welke testgevallen (*testcases*) gebruik je best? Met hoeveel testcases heb je normaal gezien voldoende?
Vervang nu het sleutelwoord `if` door `while`, bewaar het bestand en voer het programma opnieuw uit. Wat merk je op? Voor welke testcases merk je een verschil, voor welke testcases is er geen verschil?

22 Input afgesloten door stopwaarde

Schrijf een programma dat de gebruiker telkens een woord vraagt. Als de gebruiker wil stoppen, moet die het woord STOP intikken (met hoofdletters). Het programma schrijft uit hoeveel woorden er ingelezen werden (STOP niet meegeteld).

Een mogelijk programmaverloop:

```
Geef een woord: ga
Geef een woord: je
Geef een woord: mee
Geef een woord: verdwalen
Geef een woord: ik
Geef een woord: weet
```

```
Geef een woord: de
Geef een woord: weg
Geef een woord: stop
Geef een woord: STOP
9
```

Voor gevorderden: ga er vanuit dat de gebruiker altijd één woord per keer opgeeft. Zoniet, schrijf dan het aantal ingelezen lijnen uit, dan hoeft de code niet aangepast te worden.

23 Plaats van code ín de lus

Gegeven de opgave:

> *Schrijf een programma dat de gebruiker getallen opvraagt. Zolang het getal kleiner is dan 100, wordt het bijgeteld bij de som. Als het getal gelijk of groter is dan 100, dan stopt het programma met getallen opvragen. De som van de ingegeven getallen wordt uitgeschreven. (Het laatst ingegeven getal telt uiteraard niet meer mee, dat fungeert enkel als stopsignaal.)*

Gegeven dit voorstel voor oplossing:

```
getal = int(input("Geef een getal: "))
som = getal
while getal < 100:
    getal = int(input("Geef een getal: "))
    som += getal
print(som)
```

Test grondig uit met getallen die je ook zelf makkelijk kan sommeren. Wat gaat er fout en waarom? Verbeter de code.

24 Controle op input van gebruiker

Als de gebruiker input opgeeft voor een programma, gingen we er tot nu toe van uit dat die dat plichtsgetrouw en juist doet. Als er een getal gevraagd wordt en de gebruiker geeft een stuk tekst op dat niet gelezen kan worden als een getal, dan mag het programma crashen; de gebruiker moest maar doen wat gevraagd werd.

Uiteraard zullen échte programma's niet zomaar mogen crashen. Er moeten voldoende controles en veiligheidsmechanismes ingebouwd worden, zodat programma's *fool proof* zijn: zodat ze niet verkeerd gebruikt kunnen worden.

Voorlopig kunnen we nog geen controles inbouwen op het *soort* input dat de gebruiker geeft (een getal of een stuk tekst). Dus als er een getal gevraagd wordt, gaan we er nog altijd van uit dat er een getal gegeven wordt.

Maar wat als er een *even* getal gevraagd wordt, en de gebruiker geeft een *oneven* getal? Het kan gebruiksvriendelijker dan een crash van het programma. De gebruiker kan opnieuw gevraagd worden om een geldig getal in te geven.

Schrijf een programma dat de gebruiker om een even getal vraagt. Daarna wordt het tienvoud van dat getal uitgeschreven. (Tip: een getal is even als de rest bij deling door 2 gelijk is aan 0.)

Test het programma verschillende keren uit. Doe achtereenvolgens alsof je

- een brave, oplettende gebruiker bent;
- een verstrooide gebruiker bent die zijn fout meteen rechtzet;
- een zeer verstrooide of moedwillige gebruiker bent die volhardt in de boosheid (en fouten).

25 Keuzestructuur voorafgegaan door controle op input

Herneem de oefening waarbij de gebruiker moest opgeven of hij M, V of X is – om dan met de juiste aanspreking verwelkomd te worden.
In de laatste versie kon het nog voorkomen dat de gebruiker geen van de zes mogelijke keuzes (M, m, V, v, X, x) opgaf. Zorg er nu voor dat het programma blijft doorvragen tot de gebruiker wel een keuze maakt. Verandert er ook iets aan de keuzestructuur die er al stond?

26 Debugtips

Hieronder vind je vijf problemen die door medeleerlingen gemeld worden. Zonder de code te bekijken, welke hint zou je hen geven zodat ze zelf de fout kunnen vinden? Kies telkens enkel de meest voor de hand liggende opdracht.

(1) Als ik het programma uitvoer, dan gebeurt er niets.
(2) Het programma zet heel snel achter elkaar heel veel gegevens op het scherm – en houdt niet op. Ik kan ook niet goed lezen wat er staat, want op de even regels staat er een getal en op de oneven regels een stukje tekst. En dat schiet allemaal door elkaar.
(3) De uitkomst van de ingewikkelde berekening blijkt niet te zijn wat er volgens de opgave moet komen. Maakte ik een tikfout?
(4) Het laatste geval van de while-lus wordt niet juist afgehandeld.
(5) Ik heb heel veel code geschreven en ben pas daarna beginnen testen. Nu vind ik de fout niet – geen idee in welk stuk het verkeerd loopt.

(a) Als er in een lus informatie uitgeschreven moet worden, dan zorg je er best voor dat elke doorloop van de lus exact één regel uitschrijft (ook als er verschillende variabelen uitgeschreven moeten worden). Als de lus dan oneindig lang doorloopt, zal je toch redelijk kunnen lezen hoe de output evolueert (altijd hetzelfde getal, of stijgende getallen,...).
(b) Zet delen van de code in commentaar. En test de volgende keer wat sneller uit. Dat heet *incrementeel werken*: stukjes code uittesten voor je aan een volgend (daarop steunend) stukje code begint te schrijven.
(c) Controleer of de structuur van de while-lus goed is: wat is het item waarop getest wordt? Waar staat deel 4 van de lus? Als dat deel niet helemaal op het einde van de body van de while-lus staat, staat het niet op zijn plaats.
(d) Allicht zit het programma in een oneindige lus vast (en gebeurt er dus net *heel veel*). Schrijf iets uit in de lus, of ga met een debugger / tutor na wat er in de lus gebeurt. En allicht ben je gewoon deel 4 van de lus vergeten, dus hoef je zelfs niet te gaan debuggen als dat eerst rechtgezet wordt.
(e) Voor de tigste keer de formule in codevorm herlezen zal geen zoden aan de dijk zetten. Neem pen en papier, schrijf heel bewust de formule die in de code staat over, maar zet ze meteen in wiskundige vorm. Dus `(a+b)/c` wordt $\frac{a+b}{c}$. Schrijf ook over wat er werkelijk staat, niet wat je hoopt er te vinden. Allicht kom je dan zo op de fout uit.

27 Herhaling waarbij aantal gekend is

In een van de eerste oefeningen werd er `hip hip hip hoera` op het scherm geschreven. Herschrijf deze oefening, met deze drie veranderingen:

- er wordt geen vermenigvuldiging gebruikt (dus nergens een *-teken)
- er wordt 17 keer `hip` uitgeschreven (de jarige werd net 17)
- de tekst moet niet eerst bewaard worden in een variabele; elk woord mag direct uitgeschreven worden, en dat mag telkens op een nieuwe regel

Controleer nadien grondig. Heb je écht 17 keer `hip` staan? Hoe zou je kunnen nagaan of de code zeker juist is, zonder je te mispakken aan de telling van die 17 woorden?

28 Herhaling waarbij teller gebruikt wordt

Schrijf een programma dat de gebruiker een onder- en bovengrens vraagt. Als de ondergrens groter is dan de bovengrens, dan wisselt het programma die zelf zonder tussenkomst van de gebruiker. Daarna worden alle opeenvolgende getallen vanaf de ondergrens tot en met de bovengrens uitgeschreven, in stappen van 10.

Een voorbeeld van het programmaverloop:

```
geef ondergrens: -10
geef bovengrens: 35
-10
0
10
20
30
```

Een ander voorbeeld van het programmaverloop:

```
geef ondergrens: 70
geef bovengrens: 55
55
65
```

En ook dit zou moeten werken:

```
geef ondergrens: 20
geef bovengrens: 20
20
```

29 Sommeren

Schrijf een programma dat de gebruiker eerst opvraagt hoeveel (reële) getallen die wil ingeven. Daarna worden er exact zoveel getallen opgevraagd, en allemaal gesommeerd. De som wordt uitgeschreven.

Een mogelijk programmaverloop:

```
Hoeveel getallen wil je opgeven? 3
Geef getal: 1.1
Geef getal: 2.2
```

```
Geef getal: 3.3
6.6
```

30 Wanneer welke lus?

Schrijf een programma dat de 1e tot en met de 10e macht van 2 uitschrijft. (Dit kan vrij eenvoudig.)

Schrijf daarna een programma dat opnieuw 10 getallen uitschrijft te beginnen met het getal 2, maar nu is het *volgende* getal altijd het kwadraat van het *vorige*. Dat geeft *niet* dezelfde getallen als hierboven!

Schrijf tot slot een programma dat getallen uitschrijft te beginnen met het getal 2, waarbij het volgende getal nog altijd het kwadraat van het vorige is, maar waarbij de getallen allemaal kleiner blijven dan 100 000 (de lus stopt dus zodra deze bovengrens bereikt wordt).

31 De opgave bepaalt de soort lus

Bij elk probleem waar zich een 'herhaling' voordoet, is een lus aangewezen.
Een while-lus wordt gebruikt als het niet op voorhand duidelijk is hoe dikwijls de herhaling uitgevoerd moet worden. Een for-lus wordt gebruikt als dat wel op voorhand duidelijk is. Geef aan welke probleemstellingen een while-lus vragen, welke een for-lus.

(a) We maken bearnaisesaus. Er gaan vijf eierdooiers in de saus: vijf keer eentje.
(b) Daarna gaan er verschillende klontjes boter in, tot de saus dik genoeg is.
(c) Voor elke leerling in de klas bracht de Sint een zakje snoep.
(d) Een boek kaarten wordt kaart per kaart uitgedeeld aan de vier spelers.
(e) Een boek kaarten wordt kaart per kaart omgedraaid tot harten dame gevonden wordt.
(f) De gebruiker moet een positief getal ingeven. Als hij een negatief getal ingeeft, krijgt hij een nieuwe kans (en desnoods nog eens en nog eens...).
(g) Van dat positieve getal (zie (f)) wordt de priemontbinding berekend (de kleinste deler wordt herhaaldelijk uitgedeeld, tot het getal niet meer deelbaar is, bvb. $80 = 2^5 * 5$).
(h) De gebruiker geeft op hoeveel regels hij uit een bestand wil kunnen lezen. Daarna worden die regels uit het bestand gehaald en op het scherm geprint.
(i) Alle woorden uit een bestand worden ingelezen, tot er eentje gevonden wordt dat meer dan 15 letters heeft.

32 Dubbele voorwaarden

Herneem deze oefening:

> Schrijf tot slot een programma dat getallen uitschrijft te beginnen met het getal 2, waarbij het volgende getal nog altijd het kwadraat van het vorige is, maar waarbij de getallen allemaal kleiner blijven dan 100 000 (de lus stopt dus zodra deze bovengrens bereikt wordt).

Voer het programma opnieuw uit, maar start nu bij het getal 0.5 in plaats van 2. Wat gebeurt er? Kan je dit verklaren?

Los dit op, door een grens te zetten op het aantal keer dat de lus doorlopen wordt. Er mogen maximaal 10 getallen uitgeschreven worden.

33 Lussen combineren

Schrijf een programma dat de gebruiker om een geheel getal vraagt. Zolang dit getal kleiner is dan 3, wordt het opnieuw gevraagd. Daarna wordt er vanaf dit getal naar beneden afgeteld, en tot slot START uitgeschreven.

Welke herhalingen zitten er in dit programma? Weet het programma op voorhand hoe dikwijls die herhalingen uitgevoerd moeten worden? Laat dat de leidraad zijn om de juiste lussen te kiezen.

Een mogelijk programmaverloop:

```
Geef een getal >= 3: -1
Geef een getal >= 3: 0
Geef een getal >= 3: 4
4
3
2
1
START
```

Zijn deze herhalingen *genest* (zitten ze *in* elkaar), of komen ze sequentieel *na* elkaar?

34 While-lus opkuisen

Gegeven de opdracht:

> Schrijf een programma dat de gebruiker een getal tussen 1 en 10 toont (grenzen inbegrepen), en dan vraagt of het volgende getal (dat het programma klaar heeft staan) hoger of lager zal zijn. Zolang de gebruiker het juiste antwoord raadt, toont het programma het getal, zet het een nieuw getal klaar en mag de gebruiker opnieuw raden. Raadt de gebruiker fout (zal sowieso het geval zijn als het nieuwe getal per ongeluk hetzelfde is als het vorige), dan is het spel gedaan.
>
> Een voorbeeld van het programmaverloop:

```
het startgetal is
4
Zal het volgende getal hoger of lager zijn? hoooooooger
Antwoord met hoger of lager aub: hooooger
Antwoord met hoger of lager aub: hoger
JUIST, volgend getal is
9
Zal het volgende getal hoger of lager zijn? lager
JUIST, volgend getal is
3
Zal het volgende getal hoger of lager zijn? hoger
FOUT, jammer volgende getal was
3
Spel is gedaan...
```

Deze code is een werkende oplossing, maar is zwaar op de hand. Hoe kan het opgekuist worden?

```
1 import random
2
3 # onthoud twee willekeurige getallen tussen 1 en 10 (grenzen inbegrepen)
4 vorig = random.randint(1,10)
5 volgend = random.randint(1,10)
6
```

```
 7 # print het eerste getal uit
 8 print("het startgetal is ")
 9 print(vorig)
10 ok = True
11 antwoord = input("Zal het volgende getal hoger of lager zijn? ")
12
13 # stap de lus binnen met 'voorlopige' waarde
14 while ok:
15     antwoordGeldig = True
16     if antwoord == "hoger" or antwoord == "HOGER":
17         if volgend > vorig:
18             print("JUIST, volgend getal is")
19             print(volgend)
20         else:
21             print("FOUT, jammer volgende getal was")
22             print(volgend)
23             ok = False
24     elif antwoord == "lager" or antwoord == "LAGER":
25         if volgend < vorig:
26             print("JUIST, volgend getal is")
27             print(volgend)
28         else:
29             print("FOUT, jammer volgende getal was")
30             print(volgend)
31             ok = False
32     else:
33         antwoord = input("Antwoord met 'hoger' of 'lager' aub: ")
34         antwoordGeldig = False
35
36     # stel enkel een nieuwe vraag
37     # als lus mag doorgaan (ok) en antwoord geldig was
38     # (anders had je al een nieuw antwoord opgevraagd)
39     if ok and antwoordGeldig:
40         antwoord = input("Zal het volgende getal hoger of lager zijn? ")
41         vorig = volgend
42         volgend = random.randint(1,10)
43 print("Spel is gedaan...")
```

Hier volgen een aantal tips, die kunnen helpen bij het schrappen van code en helderder maken van de structuur.

a) Momenteel worden antwoorden in hoofdletters (HOGER of LAGER) ook toegelaten. Schrap dat even, dat kan later met een stringfunctie makkelijker bereikt worden (zonder een dubbele voorwaarde).
b) Momenteel wordt de gebruiker nog terechtgewezen als hij een fout woord intikt. Schrap dat even, dat passen we helemaal op het einde aan. Want zoals het er nu staat, verstoort het de hoofdstructuur. Eerst verzorgen we die hoofdstructuur.
c) Controleer alles wat er in de belangrijkste lus staat (die start op regel 14). Zal alles wat in die lus staat, ook meerdere keren uitgevoerd moeten worden? Het gaat dan om (volg in de code): het antwoord controleren / 'JUIST' uitprinten / 'FOUT' uitprinten / nieuw antwoord opvragen. Alles wat niet herhaaldelijk moet gebeuren maar slechts één keer, hoort niet *in* de lus te staan.
d) Nu wordt de belangrijkste lus (regel 14) nog binnengestapt met een 'voorlopige' logische variabele (met naam ok). Nadeel: om het verloop van de lus goed te krijgen, moet die voorwaarde onderaan (regel 39) nogmaals getest worden. Dus ze wordt twee keer per doorloop van de lus gecontroleerd.

e) Het mag uit vorige twee vaststellingen duidelijk zijn dat er wat schort aan de structuur van de belangrijkste lus. Daarom beginnen we from scratch. Dan hoeven we de slechte beslissingen niet meer mee te slepen en om te buigen.
   Formuleer een heel concreet antwoord op volgende vraag, en in het Nederlands:
   "Hoe lang gaat het spelletje door?"
   Het antwoord zou moeten klinken als
   "Zolang de gebruiker 'lager' tikt als het getal inderdaad lager is
   (of 'hoger' als het getal inderdaad hoger is)."
   Dat kan met de juiste logische operatoren (`and` en `or`) rechtstreeks in code omgezet worden. Daarmee heb je deel 2 van de while-lus.
   Duid voor jezelf duidelijk aan welke variabelen hier een rol spelen: op welke variabelen wordt er getest? Die zijn belangrijk in het volgende stuk.
f) Zet de variabelen die je zojuist geïdentificeerd hebt, klaar VOOR de while-lus. Dat is deel 1 van de while-lus.
g) Zet de variabelen die je zojuist geïdentificeerd hebt, ook opnieuw klaar NET VOOR HET EINDE van de while-lus. Dat is deel 4 van de while-lus.
h) Nu bestaat het programma uit een vijftiental regels, dat is al heel wat gewonnen. Het spel kan ook al gespeeld worden; alleen verkeerde input (iets anders dan `hoger` of `lager`) zal nog niet goed geïnterpreteerd worden.
i) Voeg nu een *controle op de input* toe: waar input gevraagd wordt, moet er desnoods herhaaldelijk opnieuw input gevraagd worden, tot de input enkel `hoger` of `lager` kan zijn. Gebruik eventueel stringfuncties om ook alle varianten met hoofdletters toe te laten.
   Dit voegt nog een viertal regels code toe aan het programma, maar grijpt niet in in de bestaande hoofdstructuur: de bestaande code schuift hooguit naar beneden, niet naar een andere kantlijn.

Werken met tekst

35 Strings verkennen

De komende oefeningen verkennen de mogelijkheden die Python biedt om met tekst te werken. Lees het spiekbriefje over strings, maar stop voor je aan het onderdeel *'Methodes van de klasse string'* bent. Het onderdeeltje *'in-operator'* staat linksonder op het eerste blad, lees dat ook.

Deze oefening is bewust zo opgesteld dat de shell gebruikt kan worden om het antwoord te vinden. Dat werkt sneller dan een programma schrijven en laten uitvoeren, waar de gebruiker (jij dus) ook nog eens woorden moet ingeven als input.

a) Schrijf in de shell de code die nagaat of de tekst `hommel` voorkomt in de tekst `schommel`. En komt `honing` voor in `hommel`?
b) Declareer een variabele `woord` en geef dit een waarde naar keuze (van type `str`). Vraag dan om volgende zaken in de shell te tonen:
   - de eerste letter van het woord
   - de laatste letter van het woord
   - de drie eerste letters van het woord
   - de drie laatste letters van het woord
c) Ga na of
   - het woord `bloemsuiker` alfabetisch voor het woord `bloem` komt

- het woord `mededogen` alfabetisch voor `vertrouwen` komt
- het woord `wasrek` gelijk is aan het woord `WASREK`

d) Schrijf de lengte van het woord `hottentottententententoonstelling` uit.

36 Palindroom

Een palindroom is een woord dat zowel van links naar rechts als van rechts naar links gelezen kan worden, en dan gewoon gelijk blijkt te zijn. Een paar voorbeelden: lepel, kajak, rekker, testset, rotator, parterretrap en koortsmeetsysteemstrook.

Schrijf een programma dat de gebruiker een woord vraagt, en dan uitschrijft of dit een palindroom is. Zo ja, schrijf dan ook de lengte uit.

Een mogelijk programmaverloop:

```
Geef een woord: boezemvrienden
dit is geen palindroom
```

Een ander programmaverloop:

```
Geef een woord: rotator
dit is een palindroom van lengte
7
```

37 Strings overlopen

Schrijf een programma dat de gebruiker een woord vraagt. Nadien schrijft het programma elke letter op een nieuwe lijn uit.

38 Slicing

In volle coronatijd stak dichter Stijn De Paepe de natie een hart onder de riem. Wil je de aanhef hiervan lezen, schrijf dan een programma dat start met de regel code die hieronder staat (knip en plak).

```
tekst = "AMSelacnsah  kesv lclhdhoe rersekrceeihuneew ,l h  jeseeltn pe stnlc,"\
        + "nuh iiedsdra  vneje nenv   lmgboioiesujkbd  avejailersk . am uaa ira t.r   ."
```

Deze tekst werd als volgt opgesteld: van de vier regels tekst waaruit de aanhef van Stijns aanmoediging bestaat, werd telkens één letter geknipt en geplakt. Een voorbeeld: indien de aanmoediging zou zijn

```
Geef
niet
op..
!!!!
```

Dan zou het programma starten met de code

```
tekst = "Gno!eip!ee.!ft.!"
```

Laat je programma de oorspronkelijke aanmoediging van Stijn op het scherm brengen.

39 Strings overlopen en letters verwerken

Kopieer de volgende tekst in het begin van een nieuw programma. Overloop dan de hele tekst, en plak elk karakterteken dat een kleine letter of een spatie is, aan elkaar in een nieuwe tekst. Schrijf die tekst uit – dat zou een zinnige zin moeten zijn.

Doe daarna hetzelfde met alle hoofdletters en underscores. Of kan dat tegelijk gebeuren, zodat de gegeven tekst slechts één keer overlopen moet worden?

```
rommeltje = "V@;schAi}N$ldp@_@EEN_?{aGdOden !E}:kuD_nnCOMen)%PLI@MEN:T _mKAN_IK_T/W§.:e(.EeEr," \
        +"_ o§;v(erMA§A :*d!e §NDEwN/e,%g _LverEtV,;Ee.N_l/__;l)e_n _£da__@n _,M)haA}Rz}en £,K_ TWA" \
        +"%  .?I,N  /,,;   % )  @ § @  ) :          k,.a)}hlil g;i§b;(r/an";
```

Nadien kan het programma nogmaals gerund worden, maar nu met onderstaande initialisatie. Ook dat zou twee uitspraken moeten opleveren.

```
rommeltje = "}Vd§ReO £E§eGEen;*!Rza_a(.mA)L%/S(h_e)iEdR_VIS;}%I isT E{_KWA%/een, stil" \
            +"£.@lM?e s_Gt£*%/AVEo!;rN$m: _§WE_diK}(e£ ).!O$al//F oF.I}En(z_e:_ _d;o_" \
            +"_de_ t/_a_kkN:eUn/_G )EaV:E}$N,f_WEb_,O/r%eN£Z*Ee{k_tW@)!!? IF;I, *_{CO" \
            +" D{E    *!    !.;{;}ka)h£li)?l%! gi(br!§§}.a?n";
```

40 Huiswerk tegen Lesdag 2

Ga online op zoek naar tekstuele informatie die aansluit bij de leefwereld van je leerlingen. Dat kan gaan om muzieklijstjes, de volledige tekst van het toneelstuk dat op het schoolfeest uitgevoerd wordt, open data van de website van de stad waar de school ligt, de namen van alle afgestudeerden sedert het ontstaan van de school, inspirerende quotes waarmee je de lessen graag doorspekt,... Download deze informatie in een bestand (tekstbestand, excel-file, csv-file,...) en bedenk zelf al eens wat je hier graag mee zou (laten) doen.

Basiskennis programmeren in Python Lesdag 2

FUNCTIES

Een aantal 'gekende' functies (zoals wiskundige functies) werden al herhaaldelijk gebruikt in de oefeningen van lesdag 1. Dus het *gebruik van* (of het *oproepen van*) functies is in principe al gekend. In deze reeks wordt ook het schrijven (of *implementeren*) van functies ingeoefend.

101 Gebruik van functies herkennen

Hieronder een kort programma. Duid aan:

1. de hoofding van de functie (dat is één regel)
2. de implementatie van de functie (dat is minstens één regel)
3. de waarde die de functie teruggeeft (van welk type is deze waarde?)
4. het hoofdprogramma
5. de plaats waar de functie opgeroepen wordt (mogelijks meerdere keren)
6. (de naam van) de parameter van de functie
7. de verschillende argumenten die meegegeven worden bij oproep van de functie

```
def is_even(getal):
    return getal % 2 == 0

a = int(input("geef een getal: "))
b = int(input("geef nog een getal: "))
if is_even(a) and is_even(b):
    print("beide getallen zijn even")
elif is_even(a+b):
    print("beide getallen zijn oneven")
else:
    print("één van beide is even, het andere is oneven")
```

102 Functies definiëren en schetsen

Hieronder code van een hoofdprogramma.

- Duid alle functies aan die gebruikt worden. In het groen komen de functies die Python zelf al voorziet, in het rood komen de functies die door de programmeur van dit hoofdprogramma gegeven moeten zijn.

```
# DEEL 1
woord = input("Geef een woord: ")
if heeft_klinkers(woord):
    print(f"{woord} bevat klinkers")
else:
    print(f"{woord} bevat geen klinkers")

# DEEL 2
aantal = aantal_klinkers("papegaaienei")
print(f"'papegaaienei' bevat {len('papegaaienei')} karakters waarvan {aantal}
klinkers")
```

```
# DEEL 3
aantal_o = aantal_letters("portretfoto","o")
print(f"'portretfoto' bevat {aantal_o} keer de letter 'o'")

# DEEL 4
if is_korter_dan_andere_tekst("kubus","balk"):
    print("'kubus' is korter dan 'balk'")
else:
    print("'kubus' is niet korter dan 'balk'")

if is_korter_dan_lengte("kubus", 4):
    print(f"'kubus' heeft minder dan 4 karaktertekens")
else:
    print(f"'kubus' heeft 4 karakertekens of meer")
```

- Geef daarna de hoofding van de functies die in het rood aangeduid zijn, en schrijf in commentaar wat de returnwaarde van de functie is (welk type, en wat het resultaat voorstelt).
  Een voorbeeld:
  voor de aangeduide functie-oproep `aantal_klinkers("papegaaienei")` komt er

  ```
  # geeft een int terug; het aantal klinkers dat in de gegeven tekst zit
  def aantal_klinkers(tekst):
  ```

- Omdat we de functies nog niet willen implementeren, maar al wel willen gebruiken (zodat het hoofdprogramma eens kan lopen), zullen we de functies *schetsen*. Dat wil zeggen dat we ze een hardgecodeerde, zelfgekozen returnwaarde laten teruggeven. Deze returnwaarde is dus niet afhankelijk van de parameters in de parameterlijst van de functie.
  Doe dit: schets elke functie, dus vervolledig de hoofding met een hardgecodeerde return-opdracht (bijv.

  ```
  return 7
  ```

  als er een getal teruggegeven moet worden).

- In deel 2 komt het woord `"papegaaienei"` drie keer voor. Help de programmeur: bewaar dit woord in een variabele, zodat de programmeur later kan hertesten met een ander woord, zonder drie keer het woord `"papegaaienei"` te moeten vervangen.

- Breid nu deel 3 uit: schrijf uit hoe dikwijls het woord `"avondrood"` de letter `"d"` bevat. (Omdat de functies nog allemaal geschetst zijn, zal je natuurlijk hetzelfde antwoord krijgen als op de vraag hoe dikwijls het woord `"portretfoto"` de letter `"o"` bevat, maar dat is niet erg.)

103 Functies schrijven: Romeinse cijfers

Ons huidig, tiendelig talstelsel is een positioneel stelsel: de positie van een cijfer in een getal bepaalt de bijdrage van dat cijfer aan het getal. Zo is de 2 in 2068 veel meer waard dan de 2 in 8602. Daarbij is een voorstelling van het begrip 'nul' onontbeerlijk. Een expliciete afbeelding voor het begrip 'nul' werd eerst door de Indiërs gebruikt; daar wordt een punt gebruikt in plaats van een cirkel/ovaaltje. Hoe oud dit gebruik is, werd onlangs door onderzoek nog herbepaald:
https://www.newscientist.com/article/2147450-history-of-zero-pushed-back-500-years-by-ancient-indian-text/

De Romeinen moesten het zonder begrip voor de hoeveelheid 'nul' stellen, en werkten een andere getalnotatie uit. Al is die minder handig voor cijferrekenen (het soort rekenen dat in de lagere school aangeleerd wordt voor optellingen, aftrekkingen en vermenigvuldigingen).

De Romeinen gebruikten volgende notaties:

```
I staat voor 1
V staat voor 5
X staat voor 10 (diX)
L staat voor 50
C staat voor 100 (Cent)
D staat voor 500
M staat voor 1000 (Mille)
```

Schrijf een functie `decimaal2romeins(getal)` die een gegeven getal uit bovenstaand lijstje (1, 5, 10, 50, 100, 500 of 1000) omzet naar diens overeenkomstige Romeinse notatie. Je mag er vanuit gaan dat de functie nooit met een ander getal als parameter opgeroepen zal worden.

Schrijf een functie `romeins2decimaal(letter)` die een letter uit bovenstaand lijstje (I, V, X, L, C, D of M) omzet naar het getal dat deze letter voorstelt.

Tip voor de ijverige leerkracht-programmeur: het is geen sinecure om van een Romeins getal zoals MCMXLIV de getalwaarde te bepalen (hier 1944). Wie daar wel aan begint zonder doordachte structuur, riskeert in een eindeloos straatje van gevallenonderzoek terecht te komen. Niet doen! Dergelijke code is foutgevoelig, niet onderhoudbaar, en niet opbeurend om te lezen.

104 Functies in functies: volkomen kwadraten

Gegeven volgende functie. Verklaar elke regel code. Is de naam van de functie goed gekozen?

```
import math

def is_volkomen_kwadraat(x):
    wortel = math.sqrt(x)
    wortel_int = int(wortel)
    return wortel_int * wortel_int == x
```

Schrijf nu een functie `aantal_volkomen_kwadraten_tussen(begin, eind)` die telt (en teruggeeft) hoeveel volkomen kwadraten er gelegen zijn in het halfopen interval `[begin, eind[`.

Test ook uit met een aantal parameters. Wat als `eind < begin`?

105 Functies in functies: van specifiek naar algemeen

Gegeven de functie

```
def is_even(getal):
    return getal % 2 == 0
```

Schrijf een tweede functie met naam `is_veelvoud_van(..,..)` die twee parameters heeft. Beide parameters zijn getallen. De functie bepaalt of het eerste getal een veelvoud is van het tweede getal. Deze functie is met andere woorden een veralgemening van de functie die hierboven staat.

Herschrijf nadien de functie `is_even`: gebruik de functie `is_veelvoud_van` in plaats de expliciete modulo-berekening die er nu staat.

Merk op: nu winnen we niet veel (er staan niet minder regels code), omdat de implementatie van beide functies op één regel kan. We zien zodadelijk voorbeelden waarbij code wel korter wordt door het oordeelkundig gebruik van functies.

106 Het nut van functies

Gegeven onderstaande code. Die is opgesteld volgens de regels van de kunst van de while-lus. (Duid de delen van de lus aan om je hiervan te vergewissen. Hoeveel regels code bevat elk deel?) Helaas staat er dan wel veel *duplicated code*: twee keer dezelfde code. Wat stel je voor om deze duplicated code weg te halen? (Een *voorstel* is genoeg, het hoeft niet per se uitgewerkt te worden.)

```
print("KIES UIT (tik getal in): ")
print("   1. begroeting")
print("   2. bereken som")
print("   3. bereken kwadraat")
print("   4. stop")
keuze = input()

while keuze != "4" :

    if keuze == "1":
        print("Dag gebruiker!")
    elif keuze == "2":
        a = float(input("geef een eerste getal: "))
        b = float(input("geef een tweede getal: "))
        print(a+b)
    elif keuze == "3":
        x = float(input("geef een getal: "))
        print(x * x)

    print("\nKIES UIT (tik getal in): ")
    print("   1. begroeting")
    print("   2. bereken som")
    print("   3. bereken kwadraat")
    print("   4. stop")
    keuze = input()

print("Dank voor het meespelen!")
```

TEKST

Lees de string-methodes die op het spiekbriefje staan (rechtkant van tweede bladzijde). Houd dit in de buurt, en duid er op aan welke methodes je gebruikt in de komende oefeningen.

107 Methodes op tekst

Schrijf een programma dat de gebruiker om een woord of getal vraagt.

- Als de input enkel cijfers bevat, dan wordt het dubbel ervan uitgeschreven.
- Als de input een woord voorstelt (enkel letters bevat), dan wordt er uitgeschreven of het enkel kleine, enkel hoofdletters, of een mix van beide bevat.
- Als de input niet enkel uit letters of uit cijfers bestaat, dan wordt er uitgeschreven of er ook ándere karaktertekens dan letters of cijfers voorkomen.

Laat het programma lopen met verschillende testgevallen. Een lijstje van de mogelijke antwoorden van het programma:

```
het dubbel van 78 is 156
'hoera' bestaat enkel uit kleine letters
'HALLO' bestaat enkel uit grote letters
'Mijnheer' bevat zowel kleine als grote letters
'nr2' bevat zowel cijfers als letters maar niets anders
'-123' bevat ook andere karaktertekens dan cijfers en letters
'Hoop, geloof en liefde!' bevat ook andere karaktertekens dan cijfers en letters
```

108 Tekst opsplitsen en weer samenvoegen

Gegeven een aantal woorden die naast elkaar op een lijn staan. (Ze werden hardgecodeerd als tekst, of een gebruiker tikt een regel woorden in en sluit af met enter. Al vraagt dat laatste meer tijd als er dikwijls getest moet worden of de code juist is.) Schrijf een programma dat de woorden in omgekeerde volgorde uitschrijft.
Om makkelijk uit te testen, kan je deze hardgecodeerde variabele overnemen in je code:

```
tekst = "Emerson Waldo Ralph ~ well. live and lived have you that difference some
make it have to compassionate, be to honorable, be to useful, be to is It happy. be
to not is life of purpose The"
```

109 Methodes op tekst en slicing

Vraag de gebruiker twee woorden. Bepaal het langste stuk dat ze achteraan gemeenschappelijk hebben. Gebruik daarvoor slicing en de methode `endswith()`.

Twee mogelijke outputs van het programma:

```
Gemeenschappelijk einde van 'mengelmoes' en 'assepoes' is 'oes'.

Gemeenschappelijk einde van 'vlinder' en 'vliegen' is ''.
```

110 Alliteraties

De dichter Guido Gezelle illustreerde het begrip alliteratie of stafrijm met deze zin:

"Stafrijmen zijn stapstenen waarop men steunt met de stemme."

In deze oefening zoeken we hoeveel woorden uit een zin beginnen met de beginletter van de zin. Test uit met volgende vier hardgecodeerde zinnen (copieer ze vooraan in het programma, en haal ze één voor één uit commentaar):

```
# tekst = "Stafrijmen zijn stapstenen waarop men steunt met de stemme."
# tekst = "De dunne dokter duwde de dikke dame door de draaideur."
# tekst = "Liesje leerde Lotje lopen langs de lange Lindelaan,\n maar toen Lotje
niet wou lopen,\n toen liet Liesje Lotje staan."
# tekst = "Wij willen Willem weg, wilde Willem wijzer worden, wij Willem weer."
```

Voor de laatste initialisatie van de variabele `tekst` komt er als output:

```
De tekst
'Wij willen Willem weg, wilde Willem wijzer worden, wij Willem weer.'
bevat 11 alliteraties op 11 woorden.
```

Tip: zoek niet enkel naar de startletter van de zin, maar naar de combinatie *'nieuw woord én startletter'*. En leg je idee eerst voor aan de buur.

LIJSTEN

Lees op het spiekbriefje van Python de informatie over *lijsten* en *herhaling (aantal vooraf gekend): for-lus*.

111 Lijsten overlopen en uitschrijven

Gegeven de lijst

```
woorden = ["beire","boeie","slay","smash"]
```

Schrijf een programma dat deze woorden op het scherm brengt, elk woord op een nieuwe regel. Doe dit op twee manieren: eerst met een foreach-lus (= zonder indexering), daarna met een for-lus (= met indexering).

112 Lijsten overlopen en elementen verwerken

Gegeven onderstaande lijst getallen

```
aantallen = [6,6,6,8,5,7,7,7,6,4,3,7,10,12,13,14,15,15,15,14,13,12,10,8,4,\
             6,10,14,16,18,19,20,21,22,22,22,22,22,22,21,20,19,18,16,14,10]
```

Merk op dat de regel te lang is om (leesbaar) op één lijn te krijgen. Daarom werd hij opgesplitst met het `\`-teken, zoals het hoort in Python. Teken voor elk van deze getallen een aantal sterretjes op een nieuwe lijn op het scherm. Zo zullen de eerste drie lijnen uit 6 sterretjes bestaan: `******`. Wat werd er getekend?
Vergelijk jouw code met die van je buur. Kozen jullie voor dezelfde oplossing?

113 Elementen in nieuwe lijst steken

Schrijf een hoofdprogramma dat een lijst van woorden overloopt. Alle korte woorden (korter dan 6 karaktertekens) worden in een nieuwe lijst met naam `kort` opgeslagen, alle andere woorden worden in een nieuwe lijst met naam `lang` opgeslagen. Schrijf nadien de drie lijsten uit (dat mag in één printopdracht, dat hoeft niet element per element).

Om te kunnen testen, maak je zelf een lijst van woorden aan.

114 Mediaan

Schrijf een functie die de mediaan van een gegeven lijst getallen berekent. Let op: de lijst kan een even of oneven aantal elementen bevatten. Probeer een oplossing te vinden zonder `if/else`-structuur. (Lees: gebruik de juiste formules.)
Hoeveel testcases zijn er nodig om zeker te zijn dat de functie doet wat ze moet doen?

Merk op: als er 'formules' gevonden moeten worden, dan komen die er pas na de bestudering van een aantal concrete situaties. Dus geef uw leerlingen nooit de indruk dat die bij jou zomaar uit de mouw rollen. (Zélfs al ga je er vanuit dat leerlingen de formule voor de mediaan 'zouden moeten kennen'.) Gebruik concrete voorbeelden, en leid daaruit een algemene regel (= formule) af.

115 Stringmethodes versus Listmethodes

Een string is een aaneenschakeling van letters. Een lijst waarin woorden bewaard worden, kan als een 'aaneenschakeling van woorden' beschouwd worden.
Er zijn heel wat methodes en operatoren die zowel op strings als op lijsten toegepast kunnen worden. Hieronder staat een hoofdprogramma, waarin een aantal gelijkenissen en verschillen overlopen worden. Voorspel wat de output is; schrijf die er naast. Nadien kan de code gekopieerd worden in een .py-bestand, om het antwoord te controleren.

```
zin = "Dit is een zin met zeven woorden."
woorden = ["Dit","is","een","lijst","van","losse","woorden."]

print(f"{zin=}")
print(f"{woorden=}")

print(f"{zin.find('een')=}")
print(f"{woorden.index('een')=}")

print(f"{zin.find('octrooi')=}")
print(f"{woorden.index('octrooi')=}")

print(f"{zin[2]=}")
print(f"{woorden[2]=}")

print(f"{zin[-1]=}")
print(f"{woorden[-1]=}")

print(f"{zin[:5]=}")
print(f"{woorden[:5]=}")

print(f"{zin[::-1]=}")
print(f"{woorden[::-1]=}")

print("zin (string) uitgeprint met foreach:")
for letter in zin:
    print(f"  {letter=}")
print("woorden (list) uitgeprint met foreach:")
for w in woorden:
    print(f"  {w=}")

print(f"{len(zin)=}")
print(f"{len(woorden)=}")

print(f"{2*zin=}")
print(f"{2*woorden=}")

zin_veranderd = zin.replace("zeven ","").replace("met","zonder")
print("Na weghalen van 'zeven' en vervangen van 'met' door 'zonder':")
print(f"  {zin=}\n  {zin_veranderd=}")

woorden.remove("van")     # elke '.remove' moet op nieuwe regel;
woorden.remove("losse")   # in tegenstelling tot '.replace' bij de string
locatie = woorden.index("is")
woorden[locatie] = "was"
print("Na weghalen van 'van' en 'losse', en vervangen van 'is' door 'was':")
print(f"  {woorden=}")
```

116 Frequenties

Gegeven een lijst getallen tussen 0 en 10 (grenzen inbegrepen). Gevraagd: schrijf uit hoe dikwijls elk getal voorkwam.

Als antwoord hierop wordt volgend programma voorgesteld. Er wordt ook een gegeven lijst hardgecodeerd, om te kunnen testen.

```
punten =
[6,4,8,7,8,8,8,5,6,4,6,5,10,7,7,5,6,3,2,9,7,4,6,10,5,2,7,3,0]
frequenties = [0]*10

for punt in punten:
    for i in range(10):
        if i == punt:
            frequenties[i] += 1

for i in range(10):
    print(f"frequentie van {i} is {frequenties[i]}")
```

Wat doet deze code? Werkt de code correct (= doet het programma wat het moet doen)? Werkt de code efficiënt? Pas zo nodig aan.

Uitbreiding

Als de frequenties niet per getal, maar per bereik van drie getallen gevraagd worden, wat verandert er dan in de code? De output zou er dus zo moeten uitzien:

```
frequentie van 0-2 is 3
frequentie van 3-5 is 9
frequentie van 6-8 is 14
frequentie van 9-10 is 3
```

117 Totems

Wie scouts zegt, zegt totem. Deze dierennaam wordt een scout toegekend op basis van zijn/haar karaktereigenschappen. Daarbij kan nog een adjectief komen, een voortotem. In deze oefening gaan we tegen alle zorgzame scoutstradities in: in plaats van een weloverwogen keuze te maken, worden er willekeurige combinaties van totem en voortotem gegenereerd.

Start met twee lijsten van gelijke lengte, waarin totems en voortotems bewaard worden.

Schrijf dan een programma dat deze totems en voortotems willekeurig combineert, zodat elke totem en elke voortotem slechts één keer voorkomt.

Als de lijsten geïnitialiseerd worden als

```
voortotems = ["Dromerige", "Zorgzame", "Vrolijke", "Selectieve"]
totems = ["Libelle", "Das", "Wolf", "Gems"]
```

is dit een mogelijke ouput:

```
Dromerige Das
Selectieve Gems
Zorgzame Wolf
Vrolijke Libelle
```

Zorg dat het programma zonder verdere aanpassingen blijft werken als er totems en voortotems bij komen.
Tip: een willekeurig getal in het gesloten interval `[a,b]` kan je opvragen aan de hand van de code `random.randint(a, b)` op voorwaarde dat de bibliotheek `random` geïmporteerd werd.

118 Desserts tellen

Een groep vrienden heeft afgesproken op restaurant. Gegeven een lijst met de desserts die iedereen besteld heeft. Bijvoorbeeld:

```
besteld =
["ijs","ijs","brownie","brownie","flan","ijs","flan","brownie","ijs","flan","flan"]
```

Om het niet te ingewikkeld te maken bij de bestelling, schrijf je een programma dat uit deze data volgende informatie filtert en uitschrijft:

```
ijs werd 4 x gekozen
flan werd 4x gekozen
brownie werd 3x gekozen
```

Merk op, de desserts worden omgekeerd alfabetisch opgesomd. Zorg dat dit ook zo is als er andere desserts in de lijst voorkomen.

Denk eerst na of er extra hulpvariabelen nodig zijn om dit probleem op te lossen. Wat niet toegelaten is (wegens niet onderhoudbaar en aanpasbaar voor een andere lijst desserts): drie variabelen voor respectievelijk het aantal keer ijs, het aantal keer brownie en het aantal keer flan.

Leg jouw oplossing eerst aan de buur voor, aan de hand van een schets van de gebruikte variabelen (geheugenplaatsen). Om de uitleg klaar en duidelijk te houden, vul je de geheugenplaatsen ook (gaandeweg) in met concrete gegevens.

119 Elementen tellen die aan voorwaarde voldoen

Gegeven een lijst gehele getallen, een deeltal en een bovengrens. Schrijf een functie die telt hoeveel getallen van die lijst een veelvoud zijn van het deeltal, én tegelijk kleiner of gelijk aan de bovengrens. Test uit met een zelfgeschreven hoofdprogramma, waarin je de lijst, het deeltal en de bovengrens hardgecodeerd klaarzet. Welke testgevallen (test cases) voorzie je?

Tip: schrijf eerst de hoofding van de functie, en ga dan controleren bij de buur. Zit die op hetzelfde spoor?

120 Lijsten: foreach-lus versus for-lus

Gegeven volgende opdracht:

Gegeven een lijst van lonen, voor indexering. De indexatie bedraagt 4%.
Schrijf de oorspronkelijke lonen uit, pas de lonen ín de lijst aan, en schrijf de geïndexeerde lonen uit.

De oplossing die wordt voorgesteld is de volgende:

```
lonen = [2500.0, 2800.0, 3000.0]

print(f"voor indexatie: {lonen}")

for loon in lonen:
    loon = loon * 1.04

print(f"na indexatie:   {lonen}")
```

Voor het programma uit en leg uit wat er gebeurt. Hier werd voor gewaarschuwd in de spiekbrief. Waar? Pas het programma daarna aan. Er zijn nu twee mogelijkheden:

- De lijst wordt niet ter plekke aangepast, maar er wordt een tweede lijst aangemaakt met de geïndexeerde lonen. Als de oorspronkelijke lijst uiteindelijk de geïndexeerde lonen moet bevatten, kan er op het einde een kopie gemaakt worden van de nieuw aangemaakte lijst. Let op: dit vraagt (tijdelijk) meer geheugenplaats, en sowieso meer werk dan de volgende oplossing.
- De lijst wordt overlopen met een lus die wél aanpassingen ter plekke toelaat. Zie spiekbrief.

121 Lijsten: gelijktijdig doorlopen

Gegeven een lijst lonen, en een lijst personen – elementen op overeenkomstige indices horen bij elkaar. Schrijf uit wie welk loon krijgt. Is het gebruik van een foreach-lus mogelijk? Is het gebruik van een for-lus mogelijk?

Start het programma met deze hardgecodeerde gegevens:

```
lonen = [2500.0, 2800.0, 3000.0]
personen = ["Myriam","Mehmet","Malika"]
```

122 Lijsten: berekening op basis van voorgaand of volgend element

Een woordslang is een opeenvolging woorden, waarbij elk volgend woord start met de laatste letter van het vorige woord. Ga na of een gegeven lijst woorden een woordslang is, en schrijf uit bij welke twee woorden het mis loopt áls de lijst geen woordslang bevat.

```
slang = ["leeuw","wolf","fret","toekan","nijlpaard",\
         "dingo","otter","rups","struisvogel","libelle","egel"]
slang = ["fret","toekan","nijlpaard","dingo","rups","struisvogel"]
```

Test uit met bovenstaande lijsten. Zet ze achtereenvolgens in/uit commentaar en laat het programma twee keer lopen.
De output voor beide situaties zou er als volgt kunnen uitzien:

```
Dit is een woordslang: leeuw-wolf-fret-toekan-nijlpaard-dingo-otter-
rups-struisvogel-libelle-egel

Dit is geen woordslang, het loopt mis bij dingo en rups
```

Voor de nieuwsgierige lezer: omdat er TWEE zaken gevraagd worden (antwoord True/False én twee woorden als het mis loopt), is het niet didactisch verantwoord om hier een functie van te maken. We spreken af dat een functie slechts één returnwaarde heeft. Uiteraard valt hier een mouw aan te passen. Zo zou het antwoord verpakt kunnen worden in een lijst (niet de beste optie!), of kan er afgesproken worden dat er één string teruggegeven wordt (met de boodschap 'woordslang' of de twee woorden waar het mis loopt).
Dit terzijde. Val er de leerlingen niet mee lastig, tenzij ze er expliciet zelf achter vragen.

123 Desserts uitdelen aan juiste persoon

Deze oefening is qua verhaal een vervolg op de bestelling van desserts, maar kan onafhankelijk gemaakt worden.

Als een vriendengroep op restaurant een bestelling desserts heeft doorgegeven, dan is het bij aankomst van het dessert handig om te weten wie welke keuze gemaakt heeft. Dan kan iedereen beginnen voor het ijs gesmolten is – of de brownie koud.

Schrijf een programma dat start met een lijstje van alle mogelijke desserts, en alle aanwezige personen.

```
desserts = ["ijs", "brownie", "flan"]
personen = ["Lies","Stan","Stef","Yara","Lynn","Amir","Elin"]
gekozen = []
```

Laat iedereen zijn keuze ingeven, en bewaar dat in de list `gekozen`. Het eerste deel van het programma zou dus als volgt kunnen verlopen:

```
Dag Lies, kies uit ijs / brownie / flan: flan
Dag Stan, kies uit ijs / brownie / flan: brownie
Dag Stef, kies uit ijs / brownie / flan: ij
          kies uit ijs / brownie / flan: ijkes
          kies uit ijs / brownie / flan: ijs
Dag Yara, kies uit ijs / brownie / flan: brownie
Dag Lynn, kies uit ijs / brownie / flan: flan
Dag Amir, kies uit ijs / brownie / flan: flan
Dag Elin, kies uit ijs / brownie / flan: ijs
```

Schrijf op het einde uit voor wie elk dessert is. Maak gebruik van een hulpfunctie om, gegeven alle `personen` en de `gekozen` desserts, voor een specifiek dessert, de namen terug te geven van de personen die voor dit dessert kozen. De returnwaarde van deze functie is een string van de vorm `naam_1, naam_2, naam_3` (dus een aantal persoonsnamen met een komma ertussen).

Het programma eindigt bijvoorbeeld (en overeenkomstig het programmaverloop van hierboven) dan met de output

```
ijs is voor Stef, Elin
brownie is voor Stan, Yara
flan is voor Lies, Lynn, Amir
```

Verander nu de initialisatie van de desserts en personen. Werkt alles nog naar behoren?

124 Tekst vervangen

Leen geeft een feestje (eigenlijk twee), want ze is binnenkort jarig. Ze wil een aantal vriendjes uitnodigen met volgende tekst:

> *Beste* ***naam****,*
> *je bent uitgenodig op mijn feestje!*
> *Kom je verkleed als* ***figuur****?*
> *Ik zie je* ***weekdag****!*
> *Leen*

Wie er komt, hoe ze verkleed mogen komen, en wanneer, staat in drie lijsten:

```
namen = ["Quinn","Ferre","Aisha","Aiden","Hanna"]
figuren = ["ridder","ridder","fee","prinses","fee"]
weekdagen = ["woensdag","zaterdag","woensdag","woensdag","zaterdag"]
```

Schrijf een programma dat de gegeven tekst vijf keer op het scherm print, waarbij de vetgedrukte gegevens vervangen worden.

Tip: steek de hele tekst van de uitnodiging in één lange string. Gebruik de twee karaktertekens `\n` om aan te duiden dat er een nieuwe regel moet komen. De combinatie van deze twee tekens staat voor *new line*; ze worden dus geïnterpreteerd als het begin van een nieuwe lijn.

125 Wanneer zijn lijsten nodig / nuttig?

Het is niet altijd makkelijk om meteen de juiste datastructuren te vinden voor een oefening. En van zodra het begrip 'lijst' gekend is, wordt dit soms zonder nadenken erbij gesleurd. Deze oefening vraagt geen codeerwerk, enkel een voorafbeschouwing. Er zitten ook enkele situaties in die een verzameling of woordenboek (dictionary) vragen, hoewel dit allicht niet behandeld kan worden. Bekijk deze situaties als achtergrondinformatie voor de leerkrachte, niet als oefening voor de leerlingen.

De keuze van de juiste datastructuren (enkelvoudige variabelen of lijsten) kan een programma maken of kraken. Hieronder worden elf verschillende problemen geschetst. Kies de juiste datastructure. Ga er vanuit dat er elf verschillende programma's geschreven worden, er wordt dus niet gesteund op informatie die in het vorige probleem bewaard werd.
Tip: pin een voorbeeld vast (verzin zelf een bestand met een aantal losse woorden in), en ga na hoe je zelf het gevraagde resultaat zou berekenen. Welke variabelen of geheugenplaatsen heb je nodig?

(1) Alle woorden uit een bestand worden in dezelfde volgorde op het scherm geschreven.
(2) Alle woorden uit een bestand worden in omgekeerde volgorde op het scherm geschreven.
(3) Alle woorden uit een bestand worden in alfabetische volgorde op het scherm geschreven. Dubbels worden niet genoteerd.
(4) De gebruiker van het programma kan één maal vlot nagaan of 1 bepaald woord in het bestand voorkomt.
(5) De gebruiker van het programma kan verschillende malen na elkaar vlot nagaan of een bepaald woord in het bestand voorkomt.

(6) Elk tiende woord uit een bestand wordt op het scherm geschreven (in volgorde van voorkomen).
(7) Het aantal verschillende woorden uit een bestand wordt op het scherm geschreven.
(8) Alle verschillende woorden uit een bestand worden in alfabetische volgorde op het scherm geschreven.
(9) Alle woorden uit een bestand worden in volgorde van voorkomen, maar geordend volgens lengte, op het scherm uitgeschreven. (Eerst de kortste woorden.)
(10) Van elke lengte wordt uitgeschreven hoeveel woorden van die lengte in het bestand voorkomen. Gelijke woorden worden elk apart gerekend.
(11) Van elke woord kan opgezocht worden hoe dikwijls het voorkomt.

Oplossing:

(1) Geen lijst (of set) nodig; elk woord wordt meteen na inlezen op het scherm uitgeschreven.
(2) Hier is wel een lijst (van woorden) nodig.
(3) Omdat dubbels niet genoteerd worden, kan hier best een set gebruikt worden. Is dat niet gekend, dan kan de lijst eerst gesorteerd worden (met dubbels), en tijdens het uitschrijven dan gecontroleerd worden op dubbels. Maar dat bemoeilijkt de zaken erg.
(4) Geen lijst (of set) nodig.
(5) De woorden worden best opgeslagen in een set (daarin kan het snelst gezocht worden – al merkt de gebruiker daar in kleine programma's niets van); een lijst zou ook mogelijk zijn maar daar wordt minder efficiënt in gezocht.
(6) Geen lijst (of set) nodig.
(7) Dit kan het efficiëntst opgelost worden met een set (dubbels worden er vanzelf uitgefilterd), maar het zou ook kunnen met een lijst: na inlezen van een woord test je eerst of het al in de lijst zit (daarvoor bestaan methodes, dat hoeft niet 'handmatig'); indien niet, dan wordt het toegevoegd; indien wel, dan wordt het niet toegevoegd. Maar zoeken in een lijst is, zoals gezegd, minder efficiënt dan in een set. Op het einde wordt de lengte van de lijst of set weergegeven.
(8) Wie geschoold is in Java zou hier onmiddellijk aan een TreeSet (geordende set) denken. Python heeft een analogon, maar leidt ons te ver (https://stackoverflow.com/questions/1653970/does-python-have-an-ordered-set). Alternatief: de methode van vorige oefening gebruiken, en achteraf de lijst sorteren. Niet te gebruiken als efficiëntie de topvereiste is.
(9) Gebruik een lijst van lijsten: op index 7 staat een lijst met alle woorden van lengte 7. (Dat het element op index 0 leeg blijft, is niet erg; leesbaarheid gaat hier voor op die ene verkwiste geheugenplaats.)
Een dictionary kan ook (indien gekend door de leerlingen; in Java heet dit een map); de values zijn dan lijsten.
(10) Gebruik een lijst die frequenties bijhoudt. (We gaan er vanuit dat woorden een bepaalde maximumlengte hebben, zodat de lengte van de lijst op voorhand vastligt.)
Indien echter de datastructuur dictionary gekend is, kan dit ook (vooral nuttig als de sleutels van de dictionary gehele getallen zijn die ver uit elkaar liggen).
(11) Hier is enkel een dictionary mogelijk. (Omwegen die een lijst gebruiken zijn geen goed didactisch voorbeeld, ze compliceren de zaken erg en zijn niet zo efficiënt.)

126 Lijsten mixen

Gegeven twee lijsten van tekst. Beide lijsten zijn even lang; hun lengte is bovendien even. Schrijf een functie `gemixed(een_lijst, andere_lijst)` die op basis van die twee lijsten een nieuwe lijst aanmaakt. De nieuwe lijst bevat evenveel elementen als de gegeven lijsten: de elementen op even posities (0, 2, 4,...) zijn de overeenkomstige elementen uit de ene lijst; de elementen op oneven posities (1, 3, 5,...) zijn de overeenkomstige elementen uit de andere lijst.

Test deze functie twee keer uit met onderstaande lijsten; wissel de volgorde van de parameters om de tweede testcase te verkrijgen.

```
lijst_a = ['een', 'je', 'drie', 'gehoord', 'hoedje', 'de', 'papier', '?']
lijst_b = ['heb', 'twee', 'wel', 'vier', 'van', 'van', 'zevensprong', '...']
```

Uitbreiding: testdata uit bestand

Daarna kan je diezelfde functie nogmaals uittesten met twee lijsten die de zinnen van bekende kinderliedjes bevatten. In plaats van de lijsten hardgecodeerd klaar te zetten zoals hierboven, wordt hun inhoud uit twee tekstbestanden gehaald. Download daarvoor de twee tekstbestanden paddenstoel.txt en stationnetje.txt, en zet ze in de map waar ook het huidige Python-programma staat. Voeg onderstaande code toe aan het hoofdprogramma. Dan zal `lijst_a` een lijst zijn van lengte 8, waarbij elk element een volledige zin van het bestand *paddenstoelt.txt* bevat. (Om de afsluitende enter nog weg te halen, kan je de string-methode `strip()` gebruiken.)

```
with open("paddenstoel.txt","r") as bestand_a:
    lijst_a = bestand_a.readlines()
with open("stationnetje.txt","r") as bestand_b:
    lijst_b = bestand_b.readlines()
```

Als het resultaat gezongen kan worden, is de functie `gemixed` goedgekeurd.

127 Structuur van de while-lus

Herlees de onderdelen over de while-lus en de list op het spiekblad Python. Lees daarna onderstaande programma's, en beantwoord de vragen zonder de programma's uit te voeren.

```
vogels = ["mus","mees","roek","raaf","kauw","kraai"]
i = 0
gezocht = "raaf"
while vogels[i] != gezocht and i<len(vogels):
    i += 1
if vogels[i] == gezocht:
    print(f"de gezochte vogel {gezocht} staat op index {i} in de lijst")
```

(a) Duid deel 2 van de lus aan.
(b) Duid in deel 2 alle expressies aan waarvan de waarde niet zal veranderen.
(c) Duid in deel 2 aan op welke variabele getest wordt.
(d) Duid deel 1 en deel 4 van de lus aan. Staan die op de juiste plaats?
(e) Duid deel 3 van de lus aan.
(f) Voorspel de output van het programma.
(g) Initialiseer de variabele `gezocht` nu met de waarde `"ooievaar"` in plaats van `"raaf"`. Wat is nu de output? Voer het programma uit.
(h) Pas het programma aan, en zorg dat de gebruiker sowieso een boodschap krijgt, ook als de gezochte vogel niet gevonden werd.
(i) Zoek op wat *'lazy evalution'* betekent en leg uit wat dit te maken heeft met deze oefening.

Merk op: er bestaat al een methode waarmee aan een lijst gevraagd kan worden op welke locatie een bepaald element terug te vinden is. Dus je zou kunnen opperen dat deze oefening zinloos was; gewoon de juiste methode uit de kast trekken... De getoonde expliciete werkwijze om door een lijst te lopen (en niet buiten de grenzen te vallen) blijft echter nuttig. Want misschien wordt er niet gezocht naar een lijstelement dat exact gelijk is aan een gegeven waarde, maar wel naar een woord uit de lijst dat rijmt op "grauw"*. En dan helpt de methode* `index(...)` *niet.*

128 De speld in de hooiberg

Het DNA bestaat uit een opeenvolging van vier basen: adenine, guanine, cytosine en thymine. Deze worden voorgesteld met de letter A, G, C en T.

Bij DNA-onderzoek gaat men na of een bepaalde (kortere) DNA-sequentie voorkomt in een (langere) DNA-sequentie. De kortere sequentie zullen we de speld noemen, de langere is de hooiberg. Er wordt dus gezocht naar de speld in de hooiberg.

Stel dat de hooiberg de sequentie `"ATCCGGTAGTAC"` is, en de speld is `"CCGG"`. Dan kunnen we zien dat de speld op index 2 van de hooiberg te vinden is.

Schrijf een functie `vindplaats(speld, hooiberg)` die voor een gegeven speld, op zoek gaat naar diens vindplaats (index) in de hooiberg. Als die niet gevonden wordt, dan komt er -1.

Gebruik een hulpfunctie die nagaat of de speld op een specifieke vindplaats in de hooiberg staat. (Hoeveel parameters heeft deze functie dan nodig?) Deze hulpfunctie gebruikt slicing; daardoor is de implementatie heel kort.

Test daarna uit met spelden en hooibergen van type string, bijvoorbeeld:

```
hooiberg =
"ATTCCCAACCCGGAGCAAACATATATCGGATTCTACTTGGGCTCTTTCTTTTAAGCGTAGGGCGATA
ATTTCCTACTAGAGAAGATCGTTATGAATCGATTGCTAGATGTACCGCCAAGGCTGGGTGCGACTCCT
AAGTCCTTGTATCTCTGCAGACCCTATGGGCAGCAACGGACTATAGTAACCACGGCGATCCCCTATAC
TAACGAAGGAAACGCCTTCTGCCATGAGGGAAACTACCGCGGAGGAGTCTACATCCTCGGCATGTAGG
GGCATTTTGCGGGT"
speld_1 = "TATACTAA"            # staat op index 198
speld_2 = "TATATATAAAA"         # is niet aanwezig
```

Maar ook spelden en hooibergen van type list zijn mogelijk. Dat kan, omdat strings en lijsten wel wat methodes en operatoren gemeenschappelijk hebben, met een gelijkaardige functionaliteit.

```
hooiberg =
["aap","noot","mies","wim","zus","jet","teun","vuur","gijs","lam","k
ees","bok","weide","does","hok","duif","schapen"]
speld_1 = ["jet","teun","vuur"]      # te vinden op index 5
speld_2 = ["jet","teun","roos"]      # niet aanwezig (-1)
speld_3 = ["duif","schapen"]         # te vinden op index 15
speld_4 = ["duif","schapen","stoel"] # niet aanwezig (-1)
```

Uitbreiding: hooibergdata uit bestand

Stel dat je een bestand wil doorzoeken; op zoek naar een stuk tekst (bestaande uit meerdere regels). Dan kan dit met de methode die net geschreven werd. Daarvoor moet het bestand eerst, regel per regel, in een lijst opgeslagen worden. Dat kan met onderstaande code:

```
with open("dagen.txt","r") as bestand:
    hooiberg = bestand.readlines()
```

Test dit uit: download het bestand dagen.txt, zet het in dezelfde map als het python-bestand met het programma, en schrijf de hooiberg uit. Dan komt er deze output:

```
['vrijdag\n', 'woensdag\n', 'donderdag\n', 'dinsdag\n', ...]
```

Helaas werd de *new line* mee ingelezen én bewaard. Doorloop de hele lijst, en vervang elk woord door zijn *gestripte versie* (kijk op het spiekbriefje voor de juiste methode).
Schrijf de hooiberg nadien opnieuw uit ter controle.

Kan je nu opzoeken of onderstaande spelden te vinden zijn?

```
speld_1 = ["donderdag","donderdag","woensdag","maandag","donderdag"]
speld_2 = ["maandag","dinsdag","woensdag","donderdag","vrijdag"]
```

129 Interpolatie

Gegeven het hoofdprogramma

```
waarden = [10.0, 20.0, 40.0, 100.0]
print(waarden)
print(interpolatie(waarden))
print(interpolatie(interpolatie(waarden)))
```

met bijhorende output

```
[10.0, 20.0, 40.0, 100.0]
[10.0, 15.0, 20.0, 30.0, 40.0, 70.0, 100.0]
[10.0, 12.5, 15.0, 17.5, 20.0, 25.0, 30.0, 35.0, 40.0, 55.0, 70.0, 85.0, 100.0]
```

Leid zelf af welke functie hier gebruikt wordt, en wat die functie doet. (Wat is haar naam, wat is haar parameterlijst, wat is het returntype, en hoe wordt de returnwaarde berekend uit de parameter?) Implementeer de functie, zodat jouw programma dezelfde output geeft.

130 Rijmwoorden

Voorbereiding

Gegeven de implementatie van twee functies, `aantal_klinkers(woord)` en `aantal_medeklinkers(woord)`:

```
def aantal_klinkers(woord):
    aantal = 0
    for letter in woord:
        if letter in "aeiou":
            aantal += 1
    return aantal


def aantal_medeklinkers(woord):
    aantal = 0
    for letter in woord:
        if letter in "bcdfghjklmnpqrstvwxyz":
            aantal += 1
    return aantal
```

Er kan opgemerkt worden dat hier duplicated code staat: beide functies bevatten code die heel gelijkaardig is. Enkel de constanten `"aeiou"` en `"bcdfghjklmnpqrstvwxyz"` zijn verschillend. Schrijf een derde functie

```
aantal_letters_uit(te_overlopen_woord, te_vinden_letters)
```

die telt hoeveel letters van `te_overlopen_woord` in de tekst `te_vinden_letters` zitten. Je zal merken dat je hier voor de *derde* keer duplicated code krijgt.

Test deze functie eerst uit, met parameters naar keuze.
Gebruik deze functie nadien (lees: roep deze functie op) in de implementatie van de functies `aantal_klinkers(woord)` en `aantal_medeklinkers(woord)`: op deze manier valt de duplicated code weg.

Rijmwoorden

Rijmwoorden zijn woorden die op dezelfde klank eindigen. Zo zijn *mouw* en *pauw* officieel rijmwoorden. Jammer genoeg is de schrijfwijze van de ou-klank hier niet gelijk – dat maakt het programmeren van een functie die rijmwoorden zoekt, iets lastiger. Om de oefening doenbaar te houden, definiëren we rijmwoorden nu als volgt:

> Twee woorden rijmen op elkaar als hun langste gemeenschappelijke staart gelijk is.
> Deze langste gemeenschappelijke staart mag bovendien niet leeg zijn, en moet ofwel enkel uit klinkers bestaan, ofwel minsens één medeklinker en één klinker bevatten.

Schrijf eerst een functie `langste_staart(a,b)` die van twee woorden a en b de langste staart bepaalt. Dat is de langst mogelijke deeltekst (substring) waarop a en b eindigen. Test dit uit met volgend hoofdprogramma:

```
print(langste_staart("oei","foei"))
print(langste_staart("regen","zegen"))
print(langste_staart("proberen","genereren"))
print(langste_staart("appel","boom"))
print(langste_staart("paf","puf"))
```

Schrijf een functie `zijn_rijmwoorden(a, b)` die nagaat of twee woorden a en b rijmwoorden zijn. Daarvoor controleer je nog de bijkomende voorwaarden op hun langse staart. Mogelijks is het een goed idee om nog enkele hulpmethodes te schrijven. Bijvoorbeeld een methode die nagaat of een woord minstens één klinker en één medeklinker bevat. Het voordeel om extra hulpfuncties te schrijven: als de namen daarvan goed gekozen worden, dan kunnen de voorwaarden op de langste staart, die in de opgave gegeven werden, makkelijk herkend worden in de code.

Test tot slot uit met een aantal woorden. Dat kan zoals hierboven (waarbij per test een regel code geschreven wordt), maar dat kan ook met een lus: als onderstaande lijsten samen doorlopen worden, kan er van de overeenkomstige woorden bepaald worden of ze rijmwoorden zijn. Dat zou het geval moeten zijn voor alle duo's, behalve de twee laatste.

```
woorden_a =
["nieuw","duosprong","mengelmoes","papegaai","toccata","climax","mis","gedonder"]
woorden_b =
["kieuw","pingpong","assepoes","hamerhaai","chipolata","tenorsax","poes","oplawaai"]
```

Opmerking voor de aandachtige lezer. Wie de opgave letter voor letter gevolgd heeft, en dan meer specifiek de mededeling „De langste gemeenschappelijke staart [...] moet ofwel enkel uit klinkers bestaan, ofwel minsens één medeklinker en één klinker bevatten", *heeft allicht veel geoefend op het implementeren en het schrijven van (hulp)functies. Maar wie heeft ook gezien dat die twee voorwaarden door één voorwaarde vervangen kunnen worden? En dat er dus minder (hulp)functies nodig zijn? Sowieso zijn er redelijk wat hulpfuncties geschreven die breder inzetbaar zijn dan alleen voor deze opgave – die misschien wel efficiënter kan opgelost worden zonder het hele lijstje voorgestelde hulpfuncties. Het onderwerp „efficiënte van code" komt aan bod in module 2 en 3 van de nascholing.*

131 Recursie

Deze oefening kan je enkel doen met de hele klas tegelijk. De leerkracht geeft de nodige instructies.

Voor de leerkracht: illustreer de kracht van recursie, maar ook de valkuilen aan de hand van de getallen van Fibonacci. Benodigdheden: voor elke leerling twee kaartjes met daarop de tekst 'geef me het .... getal van Fibonacci'. Meer uitleg mondeling in de nascholing.

132 Kritische zin aangescherpt

Dikwijls wordt een programma als 'afgewerkt' beschouwd als het één (of een paar) testcases met succes doorstaan heeft. Maar wat is 'een testcase'?

Gegeven een programma, geschreven door een medeleerling, dat beweert een lijst van getallen van klein naar groot te sorteren. Hoe ga je testen of dat programma inderdaad doet wat het beweert? De oorspronkelijke lijst getallen draagt de naam `gegeven`, de "gesorteerde" lijst draagt de naam `gesorteerd`. Bedenk een algoritme dat (of een werkwijze die) nagaat of de lijst `gesorteerd` inderdaad juist gesorteerd is. Het algoritme mag je beschrijven in woorden, het hoeft niet geprogrammeerd te worden.

Oplossingen

Oplossingen zijn te vinden via users.ugent.be/~lbrouns/nascholing
Of via mail naar leen.brouns@ugent.be