Algemeen

Oneindige lussen

Let op voor oneindige lussen. Een oneindige lus is een lus die eindeloos blijft uitgevoerd worden: in de meeste gevallen gaat het om een `while`-lus waarbij de statements binnen de lus er nooit voor zorgen dat de voorwaarde uiteindelijk `False` wordt. Bekijk bij wijze van bijvoorbeeld eens het volgend stuk code

```
>>> i = 0
>>> a = 0
>>> while i < 4:
...     a += 1
```

Omdat het statement `a += 1` er nooit zal voor zorgen dat de initiële waarde van de variabele `i` groter dan of gelijk aan 4 wordt, zal de voorwaarde `i < 4` altijd de waarde `True` opleveren.

Tip: Als je werkt met Pycharm, dan kan je nagaan dat je programma aan het uitvoeren is, door het rode vierkantje links van de Console of rechtsonder de menubalk. Als je op dit vierkantje klikt, dan forceer je om de uitvoering van het programma te stoppen.

Apen en kokosnoten

Tellen vanaf nul

Informatici beginnen standaard te tellen vanaf 0, niet vanaf 1, en Python volgt deze traditie op heel wat verschillende plaatsen. Zo genereert de ingebouwde functie `range` een reeks opeenvolgende getallen die begint bij 0, als je slechts één enkel argument meegeeft aan de functie. Zo tel je bijvoorbeeld standaard tot 5 in Python

```
>>> for i in range(6):
...     print(i)
...
0
1
2
3
4
5
```

Als je niet wil beginnen tellen vanaf 0, dan kan je de startwaarde meegeven als een tweede argument aan de `range` functie.

```
>>> for i in range(1, 6):
...     print(i)
...
1
2
3
4
5
```

Het wordt echter als meer *pythonic* beschouwd om het voorgaande op de volgende manier uit te schrijven

```
>>> for i in range(5):
...     print(i + 1)
...
1
2
```

```
3
4
5
```

Specifieke info

Bij het omschrijven van een aantal noten zijn er drie mogelijkheden

- geen noten
- 1 noot
- n noten

waarbij n een getal groter dan 1 is. We kunnen deze drie mogelijkheden telkens opdelen in twee componenten, die van elkaar gescheiden worden door een spatie:

- aantal noten (`geen`, 1 of n)
- enkelvouds- of meervoudsvorm van noten ('noot' of 'noten')

Volgens het **verdeel-en-heers** principe kunnen we dus best deze twee onderdelen afzonderlijk bepalen. Als we veronderstellen dat de variabele `n` verwijst naar een integer met een waarde die overeenkomt met het aantal noten, dan kunnen we als volgt de string samenstellen die het aantal noten omschrijft met de correcte enkelvouds- of meervoudsvorm:

```
>>> aantal = 'geen' if n == 0 else str(n)
>>> noten = 'noot' if n == 1 else 'noten'
>>> f'{aantal} {noten}'
```

Onderstaande voorbeelden illustreren dit voor een aantal gevallen:

```
>>> n = 0
>>> aantal = 'geen' if n == 0 else str(n)
>>> noten = 'noot' if n == 1 else 'noten'
>>> f'{aantal} {noten}'
'geen noten'
```

```
>>> n = 1
>>> aantal = 'geen' if n == 0 else str(n)
>>> noten = 'noot' if n == 1 else 'noten'
>>> f'{aantal} {noten}'
'1 noot'
```

```
>>> n = 4
>>> aantal = 'geen' if n == 0 else str(n)
>>> noten = 'noot' if n == 1 else 'noten'
>>> f'{aantal} {noten}'
'4 noten'
```

Pythagorese drietallen

Specifieke info

Een eerste aanzet voor deze opgave is het gebruik van de *brute force* techniek. Hierbij probeer je alle mogelijke waarden voor a, b en c uit en controleer je telkens of de waarden leiden tot een geldige oplossing.

Als je echter op een naïeve manier alle mogelijke oplossingen uitprobeert, dan zal je tegen de tijdslimiet aanlopen die we vooraf hebben ingesteld op Dodona. In dit geval betekent het niet noodzakelijk dat je ingediende oplossing in een oneindige lus is vastgeraakt, maar dat het te lang duurt om voor alle testgevallen de gevraagde oplossing te vinden.

Je zal dus nog enkele optimalisaties moeten doorvoeren aan de *brute force* om je oplossing efficiënter te maken, door na te denken welke gevallen je niet (meer) moet controleren omdat ze zeker niet tot een geldige oplossing zullen leiden. Vermijden om overbodige combinaties te evalueren wordt in technische termen *snoeien* genoemd. Hiervoor kan je alle informatie die je meekrijgt in de opgave goed gebruiken!