Algemeen

Testen of waarde van variabele behoort tot een vast aantal opties

De volgende voorwaardelijke opdrachten bevatten een logische fout bij het controleren of een gegeven werkdag in het weekend valt of een werkdag is. Eigenlijk is het zo dat met deze formulering de voorwaarde altijd waar zal zijn, waardoor de suggestie gewekt wordt dat het altijd weekend is.

```
>>> weekdag = 'zondag'
>>> 'weekend' if (weekdag == 'zaterdag' or 'zondag') else 'werkdag'
'weekend'

>>> weekday = 'maandag'
>>> 'weekend' if (weekday == 'zaterdag' or 'zondag') else 'werkdag'
'weekend'
```

De reden waarom het hier fout loopt is dat de voorwaarde is samengesteld uit twee deelvoorwaarden, namelijk `weekdag == 'zaterdag'` en `'zondag'`. De tweede deelvoorwaarde is altijd waar, omdat elke string waar is, behalve de lege string die niet waar is. De correcte formulering van de samengestelde voorwaarde is

```
>>> weekdag = 'zondag'
>>> 'weekend' if (weekdag == 'zaterdag' or weekdag == 'zondag') else 'werkdag'
'weekend'

>>> weekday = 'maandag'
>>> 'weekend' if (weekdag == 'zaterdag' or weekdag == 'zondag') else 'werkdag'
'werkdag'
```

Let hierbij op de herhaling van de variabelenaam in de voorwaarde.

Vermijden om meerdere `print` statements te gebruiken

Het is altijd een goed idee om ervoor te zorgen dat je programma maar één print statement bevat. Zo is het makkelijker om een overzicht te behouden van wat er allemaal uitgeschreven kan worden.

Stel bijvoorbeeld dat je volgende code hebt

```
>>> x = 3
>>> if x < 5:
...     print('kleiner dan 5')
... else:
...     print('groter dan 5')
...
'kleiner dan 5'
```

Dan herschrijf je dit beter naar

```
>>> x = 3
>>> if x < 5:
...    resultaat = 'kleiner'
... else:
...    resultaat = 'groter'
...
>>> print(f'{resultaat} dan 5')
'kleiner dan 5'
```

voorwaardelijke expressies

Als je de waarde die je aan een variabele wil toekennen laat afhangen van een bepaalde voorwaarde, dan kan je hiervoor een voorwaardelijke opdracht gebruiken. Stel bijvoorbeeld dat je werkt met een variabele `x` die verwijst naar de lengte van een persoon (in centimeter) en dat je aan de variabele `lengte` de waarde `'groot'` wil toekennen als $x > 185$ en anders de waarde `'klein'`. Met een voorwaardelijke opdracht kan je dit op de volgende manier realiseren.

```
>>> x = 100
>>> if x > 185:
...     lengte = 'groot'
... else:
...     lengte = 'klein'
...
>>> lengte
'klein'

>>> x = 190
>>> if x > 185:
...     lengte = 'groot'
... else:
...     lengte = 'klein'
...
>>> lengte
'groot'
```

Je kan hetzelfde realiseren door gebruik te maken van een *voorwaardelijke expressie* (ook een *if-else-expressie* genoemd). In tegenstelling tot een voorwaardelijke opdracht is dit dus een expressie (die een waarde oplevert) en geen statement. Hierdoor kan je een voorwaardelijke expressie zonder problemen gebruiken als rechterlid van een toekenningsopdracht. De bovenstaande interactieve sessie kan dus korter geschreven worden als

```
>>> x = 100
>>> lengte = 'groot' if x > 185 else 'klein'
>>> lengte
'klein'

>>> x = 190
>>> lengte = 'groot' if x > 185 else 'klein'
>>> lengte
'groot'
```

Controleren of een getal tussen twee grenzen ligt

In Python kan je controleren of een waarde (van een variabele) binnen een interval gelegen is door gebruik te maken van een samengestelde Booleaanse expressie. Als je bijvoorbeeld wil controleren of het getal `x` in het interval $]a, b[$ ligt, dan kan je daarvoor de volgende Booleaanse expressie gebruiken

```
>>> a < x and x < b
```

In de wiskunde zou je deze voorwaarde echter schrijven als $a < x < b$ en daarom kan je in Python evengoed de volgende verkorte expressie gebruiken.

```
>>> a < x < b
```

Hieraan is duidelijk te zien dat Guido Van Rossum, de uitvinder van Python, een opleiding in de wiskunde genoten heeft. Ook de voorwaarde $a \leq x \leq b$ waarbij de grenzen behoren tot het interval kan je op dezelfde

manier in Python schrijven als

```
>>> a <= x <= b
```

Formule 1

Getallen naar boven afronden

De `math` module bevat een functie `ceil` die kan gebruikt worden om *floating point* getallen naar boven af te ronden. Deze functie geeft de afronding terug als een integer.

```
>>> import math
>>> math.ceil(3.2)
4
>>> math.ceil(3.7)
4
```

De `math` module bevat ook de omgekeerde functie `floor` die kan gebruikt worden om *floating point* getallen naar beneden af te ronden. Voor de klassieke manier om *floating point* getallen af te ronden kan je gebruikmaken van de ingebouwde functie `round`.