Algemeen

Toekenningsopdracht vs. gelijkheidstest

In Python maak je bij een toekenningsopdracht gebruik van één enkel gelijkteken en bij een gelijkheidstest (nagaan of twee objecten dezelfde waarde hebben) van twee opeenvolgende gelijktekens. Om te testen of de variabele `x` gelijk is aan twee schrijf je dus

```
>>> if x == 2:    # correct
...     pass
```

en niet

```
>>> if x = 2:     # verkeerd
  File "<myscript.py>", line 1
    if x = 2:
         ^
SyntaxError: invalid syntax
```

Reynoldsgetal

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'
```

Blad, steen en schaar

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'
```

Seizoenen

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'
>>> # FOUT!!
>>> 'weekend' if (weekdag == 'zaterdag' or 'zondag') else 'werkdag'
'weekend'

>>> weekday = 'maandag'
>>> # FOUT!!
>>> '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.

Trilateratie

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
```

Twee floating point getallen vergelijken

Computers kunnen floating point getallen niet altijd exact voorstellen, waardoor we bij het rekenen met floating point altijd rekening moeten houden met kleine afrondingsfouten.

```
>>> 0.1 + 0.1 + 0.1 == 0.3
False
>>> 1 / 3 == 5 / 6 - 3 / 6
False
```

Bij het vergelijken van twee floating point getallen kunnen we daarom best geen gebruikmaken van de vergelijkingsoperator `==`, maar kunnen we bijvoorbeeld stellen dat twee floating point getallen gelijk zijn als hun verschil niet groter is dan een zeer kleine waarde ϵ (bijvoorbeeld 10^{-6}).

```
>>> a = 1 / 3
>>> b = 5 / 6 - 3 / 6
>>> epsilon = 10 ** -6
>>> a - epsilon < b < a + epsilon
True
```

Dit laatste kunnen we korter schrijven als

```
>>> abs(a - b) < epsilon
True
```

wat zoveel zegt als dat het verschil tussen de twee kleiner is dan de maximaal toegelaten afrondingsfout ϵ.

Darts

Nauwkeurige definitie van het getal π

Een nauwkeurige definitie van het getal π vind je terug in de `math` module.

```
>>> import math
>>> math.pi
3.141592653589793
```

Cyclometrische functies uit de `math` module

De `math` module van de Python Standard Library definieert een aantal **goniometrische functies** zoals de boogcosinusfunctie (`asin`), de boogcosinusfunctie (`acos`) en de boogtangensfunctie (`atan` of `atan2`; het verschil tussen deze twee is dat `atan2` rekening houdt met het kwadrant waarin het punt ligt). Het domein van de boogsinus en de boogcosinus is $[-1, 1]$. Dat betekent dat er enkel *floating point* getallen aan deze functies kunnen doorgegeven worden die in het interval $[-1, 1]$ liggen. Hierbij moet je goed opletten dat je door afrondingsfouten geen waarden doorgeeft die net buiten dit domein liggen. Hieronder geven we aan hoe je hier rekening mee kan houden.

```
>>> import math
>>> waarde = 1.00001
>>> math.acos(waarde)                    # boogcosinus berekenen
Traceback (most recent call last):
ValueError: math domain error
>>> waarde = max(-1.0, min(waarde, 1.0)) # garanderen dat waarde in interval [-1, 1] ligt
>>> waarde
1.0
>>> math.acos(waarde)
0.0
```