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'  # FOUT!!
'weekend'

>>> weekday = 'maandag'
>>> 'weekend' if (weekday == 'zaterdag' or 'zondag') else 'werkdag'  # FOUT!!
'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 keer de functie `print` aanroept voor elke regel die moet uitgeschreven worden. 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, wil laten 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 $a < x < b$. 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
```

Conversie van waarden naar Booleaanse waarden

In Python wordt het algemeen als een betere programmeerstijl gezien (*pythonic*) om de voorwaarde uit het codefragment

```
if x != 0:
    pass
```

kortweg te schrijven als

```
if x:
    pass
```

Dit werkt omdat bij de evaluatie van een voorwaarde van een `if` statement of een `while` statement, de expressie impliciet geconverteerd wordt naar een Booleaanse waarde. Voor de meeste gegevenstypes geldt dat alle waarden geconverteerd worden naar de Booleaanse waarde `True`, behalve één enkele waarde die naar `False` geconverteerd wordt:

- voor integers wordt enkel 0 naar `False` omgezet
- voor floats wordt enkel 0.0 naar `False` omgezet
- voor strings wordt enkel de lege string (`''`) naar `False` omgezet
- voor lijsten wordt enkel de lege lijst (`[]`) naar `False` omgezet
- ...

Je zal deze verkorte schrijfwijze van voorwaarden ook heel vaak in online voorbeelden terug vinden, dus zelfs als je het zelf leesbaarder vindt om de langere versie te gebruiken, is het nog steeds nodig om de kortere notatie te kunnen lezen om zo de online voorbeelden te begrijpen.

Merk bovendien ook nog op dat het vrij zinloos is om te schrijven

```
if gevonden == True:
    pass
```

als de variabele `gevonden` reeds naar een Booleaanse waarde verwijst. Ook in dat geval kan kortweg schrijven

```
if gevonden:
    pass
```

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
```

Niet-gedefinieerde namen van variabelen

In Python is het een fout om de waarde van een variabele op te vragen als die variabele nog niet gedefinieerd is (er is nog geen waarde toegekend aan die variabele). Dit doet zich bijvoorbeeld voor in het onderstaande codefragment

```
>>> x = 4
>>> if x < 3:
...     var = 'ok'
...
>>> print(var)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'var' is not defined
```

waar bij het `print` statement de foutmelding gemaakt wordt dat de variabele `var` niet kan teruggevonden worden in de namespace. Als je de voorgaande code goed bekijkt, dan zie je dat er enkel een waarde wordt toegekend aan de variabele `var` indien de waarde van `x` groter is dan 3 (en dat is hier duidelijk niet het geval).

Paardensprong

String opsplitsen in individuele karakters

Als je de karakters van een string van lengte n wilt toekennen aan n afzonderlijke variabelen, dan kan dit op de volgende manier.

```
>>> woord = 'ab'
>>> eerste, tweede = woord
>>> eerste
'a'
>>> tweede
'b'
>>>
>>> woord = 'abcd'
>>> eerste, tweede, derde, vierde = woord
>>> eerste
'a'
>>> tweede
'b'
>>> derde
'c'
>>> vierde
'd'
```

Dit is een voorbeeld van een meer algemene techiek die *tuple unpacking* genoemd wordt.

Letters omzetten naar hun getalwaarde

In Python heeft elk karakter een getalwaarde (`int`). Zo heeft het vraagteken (`?`) als getalwaarde 63. Deze waarde kan bekomen worden met behulp van de ingebouwde functie `ord`.

```
>>> ord('?')
63
```

Het interessante aan deze getalwaarden is dat de opeenvolgende letters van het alfabet ook opeenvolgende getalwaarden hebben. Dit geldt zowel voor kleine letters als voor hoofdletters. Zo heeft de letter `a` als getalwaarde 97, waaruit we kunnen afleiden dat de letter `b` als getalwaarde 98 moet hebben. Met die wetenschap kunnen we dus op een eenvoudige manier de positie van een letter in het alfabet bepalen.

```
>>> letter = 'd'
>>> ord(letter)
100
>>> ord('a')
97
>>> ord(letter) - ord('a') + 1   # d is de 4e letter van het alfabet
4
>>> ord('z') - ord('a') + 1      # z is de 26e letter van het alfabet
26
```

De twee torens

Voorwaardelijke expressies

De meeste programmeertalen hebben een ternaire operator die de voorwaardelijke expressie genoemd wordt. Het resultaat van een voorwaardelijke expressie hangt af van een voorwaarde `test`. Python gebruikt de volgende syntax voor voorwaardelijke expressies:

```
<expr_true> if <test> else <expr_false>
```

Let er hierbij op dat de volgorde van de voorwaarde en de expressies verschilt van andere programmeertalen, waar eerst de voorwaarde `test` geschreven wordt.

Als de voorwaarde `test` naar `True` evalueert, dan levert de expressie `expr_true` het resultaat op van de voorwaardelijke expressie. Als de voorwaarde `test` naar `False` evalueert, dan levert de expressie `expr_false` het resultaat op van de voorwaardelijke expressie.

Het resultaat van een voorwaardelijke expressie kan bijvoorbeeld toegekend worden aan een variabele `resultaat`:

```
resultaat = <expr_true> if <test> else <expr_false>
```