Lichaamstemperatuur

Nauwkeurige definitie van het getal e

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

```
>>> import math
>>> math.e
2.718281828459045
```

Mondriaan

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
```

Valsmunterij

Specifieke info

Voor de eerste en tweede weging kunnen we respectievelijk bepalen welke groep m en het hoeveelste item uit de groep n lichter weegt. Op die manier kunnen we de valse munt bepalen als de munt met volgnummer $3m + n + 1$.

Als de valse munt bij eerste weging bijvoorbeeld in de eerste groep zit (groep 1-2-3), dan zal m gelijk zijn aan 0. Als de munt in de tweede groep zit (groep 4-5-6), is m gelijk aan 1 en voor de derde groep (groep 7-8-9) is m gelijk aan 2.

De tweede weging bepaalt dan op dezelfde manier het getal n.

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 (integer). Zo heeft het vraagteken (`?`) als getalwaarde 63. Deze waarde kan bekomen worden met behulp van de ingebouwde functie `ord`.

```
>>> ord('?')
69
```

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 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
```

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'
>>> # 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.

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'
```