Algemeen

Opgemaakte tekst: stringinterpolatie

Wanneer je een string op een gecontroleerde manier wil samenstellen uit vaste en variabele tekstfragmenten, dan kan het handig zijn om gebruik te maken van stringinterpolatie (Engels: *string interpolation*). Een **geïnterpolleerde string** is een gewone string waarbij er voor het openende aanhalingsteken een letter `f` staat. Vandaar dat een geïnterpoleerde string ook een **f-string** genoemd wordt.

Een f-string fungeert als een soort template, waarin elk variabel fragment wordt aangeduid met een paar accolades (`{}`). Tussen die accolades plaats je dan een expressie waarvan de berekende waarde zal ingevuld worden op de plaats van het variabel fragment.

In onderstaand codefragment definiëren we bijvoorbeeld twee variabelen `getal1` en `getal2` waarvan we de som willen uitschrijven. We gebruiken stringinterpolatie om opgemaakte tekst uit te schrijven die bestaat uit de twee termen en het resultaat van de som van die twee termen.

```
>>> getal1 = 2
>>> getal2 = 3
>>> print(f'De som van {getal1} en {getal2} is {getal1 + getal2}.')
De som van 2 en 3 is 5.
```

Een paar accolades in een geïnterpoleerde string wordt een **plaatshouder** (Engels: *placeholder*) genoemd. Binnen een plaatshouder kan je niet alleen een expressie plaatsen, maar kan je (na een dubbelpunt) ook aangeven op welke manier het resultaat van de expressie moet opgemaakt worden voor het op de positie van de plaatshouder ingevuld wordt. Meer details over de verschillende manieren om de opmaak van een plaatshouder vast te leggen, vind je in de The Python Standard Library.

Hoe controleert Dodona floating point getallen?

Indien een opgave aangeeft dat er een *floating point* getal moet uitgeschreven worden, zonder een expliciet vermelding over het aantal cijfers waarmee het getal moet uitgeschreven worden (zonder afronden of afkappen), dan zal de Dodona omgeving standaard nagaan dat het uitgeschreven getal correct is tot op zes cijfers na de komma. In principe maakt het dan dus niet uit hoeveel cijfers na de komma er precies uitgeschreven worden.

Examens verbeteren

Floats uitschrijven met een vast aantal decimale cijfers (afgerond)

Standaard schrijft de ingebouwde functie `print` floating point getallen uit met een hele reeks decimale cijfers. Soms wil je floating point getallen echter laten uitschrijven met een vast aantal decimale cijfers. Je zou ervoor kunnen opteren om gebruik te maken van de ingebouwde functie `round` die je toelaat om *floating point* getallen af te ronden tot een gegeven aantal decimale cijfers.

```
>>> print(1 / 3)
0.3333333333333333
>>> print(round(1 / 3, 2))
0.33
```

Het probleem met deze oplossing is dat er door afrondingsfouten bij de interne voorstelling van floating point getallen, toch getallen kunnen ontstaan die bij het uitschrijven niet het gewenste aantal decimale cijfers hebben.

Een betere oplossing bestaat erin om de tekst die moet uitgeschreven worden, op te maken aan de hand van stringinterpolatie. Binnen het paar accolades dat gebruikt wordt als plaatshouder in de formateerstring kan je immers opgeven hoe de ingevulde waarde precies moet opgemaakt worden. Dit doe je door na de expressie tussen de accolades een zogenaamde *format specifier* te plaatsen, die wordt voorafgegaan door een dubbelpunt (`:`).

Om een waarde uit te schrijven als een floating point getal met een vast aantal decimale cijfers gebruik je de *format specifier* `:.nf`. Hierbij staat de letter `f` voor het opmaken van een floating point getal, en het getal n voor het aantal cijfers na de komma. Hierbij wordt afronding gebruikt om de decimale cijfers te bepalen. Onderstaande code illustreert bijvoorbeeld hoe je een getal kan uitschrijven, afgerond tot op twee cijfers na de komma.

```
>>> waarde = 1/3
>>> print(f'{waarde:.2f}')
0.33
```

Voor meer details over het gebruik van *format specifiers* verwijzen we naar de The Python Standard Library.

Vermenigvuldiging: operator *

Vergeet niet om de operator `*` te gebruiken om twee getallen met elkaar te vermenigvuldigen. In de wiskunde wordt de vermenigvuldiging zonder operator genoteerd, en staat xy bijvoorbeeld voor het product van x en y. In Python kan je de vermenigvuldiging niet stilzwijgend uitvoeren.

```
>>> x = 6
>>> y = 7
>>> x * y
42
>>> xy
Traceback (most recent call last):
NameError: name 'xy' is not defined
```

The pudding guy

Reële deling versus gehele deling

Python maakt onderscheid tussen de reële deling (aangeduid door de operator `/`) en de gehele deling (aangeduid door de operator `//`). Bij reële deling is het resultaat altijd een `float`. Bij gehele deling hangt het gegevenstype van het resultaat of van het gegevenstype van de operandi. Als beide operandi integers zijn, dan is het resultaat ook een integer. Als één of beide operandi `floats` zijn, dan is het resultaat ook een `float`.

```
>>> x = 8
>>> y = 3
>>> z = 4
>>> x / y            # reële deling van twee integers
2.6666666666666665
>>> x // y           # gehele deling van twee integers
2
>>> float(x) // y    # gehele deling van een float en een integer
2.0
>>> x / z            # reële deling van twee integers
2.0
>>> x // z           # gehele deling van twee integers
2
```

Python beslist enkel en alleen maar op basis van de gebruikte operator om een reële of een gehele deling uit te voeren. Deze keuze hangt dus niet af van het gegevenstype van de operandi.

```
>>> x = 7.3
>>> y = 2
>>> x // y
3.0
>>> y // x
```

```
0.0
>>> x / y
3.65
```

Ontwikkelingsindex

Extra wiskundige functionaliteit: de `math` module

De programmeertaal Python wordt bewust zo minimaal mogelijk gehouden. Er zijn echter mechanismen in de programmeertaal ingebouwd waarmee nieuwe functionaliteit aan de taal kan toegevoegd worden. Als je Python installeert, worden er een aantal modules met bijkomende functionaliteit meegeleverd. Deze modules worden samen The Python Standard Library genoemd.

De `math` module is één van die modules uit The Python Standard Library. Zoals de naam al doet vermoeden, voegt die heel wat extra wiskundige functionaliteit toe aan Python. Voor je de functionaliteit uit een module kunt aanspreken, moet je die module eerst importeren. Hiervoor bestaan er twee manieren.

Een eerste manier bestaat erin om de module op zijn geheel te importeren. Nadat je dit gedaan hebt, moet je namen van variabelen, functies of klassen uit de module laten voorafgaan door de naam van de module en een punt als je ze wilt gebruiken in je Python code.

```
>>> import math
>>> math.sqrt(16)          # vierkantswortel
4.0
>>> math.log(100)          # natuurlijke logaritme
4.605170185988092
>>> math.log(100, 10)      # log10
2.0
>>> math.pi                # nauwkeurige waarde van pi
3.141592653589793
```

Een tweede manier bestaat erin om de namen van variabelen, functies of klassen uit de module rechtstreeks te importeren in je Python code. Hierna kan je deze namen gebruiken zonder ze te laten voorafgaan door een extra prefix.

```
>>> from math import sqrt, log, pi
>>> sqrt(16)               # vierkantswortel
4.0
>>> log(100)               # natuurlijke logaritme
4.605170185988092
>>> log(100, 10)           # log10
2.0
>>> pi                     # nauwkeurige waarde van pi
3.141592653589793
```

Voor een volledig overzicht van de variabelen en functies die gedefinieerd worden in de `math` module, verwijzen we naar The Python Standard Library.

Vierkantswortel

Om de vierkantswortel van een getal te berekenen, kan je gebruikmaken van de functie `sqrt` uit de `math` module.

```
>>> import math
>>> math.sqrt(121)
11.0
```

```
>>> math.sqrt(1234)
35.12833614050059
```

Omdat $\sqrt{x} = x^{1/2}$ kan je ook gebruikmaken van de machtsverheffing (operator `**`).

```
>>> 121 ** (1 / 2)
11.0
>>> 1234 ** 0.5
35.12833614050059
```

Op dezelfde manier kunnen de derde, vierde, ... machtswortel als volgt berekend worden

```
>>> 27 ** (1 / 3)
3.0
>>> 22 ** (1 / 4)
2.1657367706679937
```

Specifieke info

Als hulp bij het debuggen, vind je hieronder een aantal tussenliggende waarden die moeten berekend worden vooraleer je het antwoord kan uitschrijven dat hoort bij de gegeven voorbeeldinvoer:

$$\begin{aligned} \text{LEI} &= 0.9718780096308187 \\ \text{EI} &= 0.846147794929596 \\ \text{MYSI} &= 0.8233902000000001 \\ \text{EYSI} &= 0.7864077669902911 \\ \text{II} &= 0.8562070881051073 \end{aligned}$$

De gestopte klok

Rest na gehele deling: gebruik van de module operator

In Python kan je gebruikmaken van de modulo operator (`%`) om de rest te bepalen na een gehele deling. Als beide operandi integers zijn, dan is het resulaat zelf ook een integer. Van zodra één van beide operandi een `float` is, dan is het resultaat zelf ook een `float`.

```
>>> 83 % 10
3
>>> 83.0 % 10
3.0
>>> 83 % 10.0
3.0
>>> 83.0 % 10.0
3.0
```

Specifieke info

Voor deze oefening is het een goed idee om alle tijdstippen op een 24-uursklok om te zetten naar het aantal minuten dat verstreken is sinds middernacht. Als het tijdstip $u : m$ is, dan is het aantal minuten dat verstreken is sinds middernacht gelijk aan

$$60 \times u + m$$

Veronderstel dat het aantal minuten dat verstreken is sinds middernacht gelijk is aan t, dan kunnen we het bovenstaande proces omkeren om daaruit terug de tijd $u : m$ op een 24-uursklok af te leiden. In Python kan je dit op de volgende manier implementeren:

```
>>> verstreken = 868               # aantal minuten verstreken sinds middernacht
>>> uren = (verstreken // 60) % 24 # aantal uren verstreken sinds middernacht
>>> minuten = verstreken % 60      # aantal minuten verstreken sinds laatste uur

>>> uren
14
>>> minuten
28
```

Let hierbij op het feit dat we gebruikmaken van gehele deling (`//`) en rest na gehele deling (`%`). Het gebruik van de modulo operator bij het bepalen van het aantal uren dat verstreken is sinds middernacht (`% 24`) zorgt ervoor dat het uur op een 24-uursklok nog steeds correct bepaald wordt als het aantal minuten dat verstreken is sinds middernacht langer is dan een volledig dag.

Op basis van bovenstaande methode kunnen we nu het aantal minuten berekenen dat Andrea van thuis weg is, evenals het aantal minuten dat ze bij haar vriending was. Hierbij moet je er alleen goed op letten dat de twee tijdstippen waartussen je de tijdsduur wil bepalen niet in dezelfde dag moeten vallen (ze kan bijvoorbeeld voor middernacht vertrekken en pas na middernacht terug naar huis gaan).