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 stringinterpolatie (Engels: *string interpolation*) te gebruiken. 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 omgezet worden naar een string (`str`) die ingevuld wordt 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 Python Standard Library.

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 kiezen om de ingebouwde functie `round` te gebruiken, 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 f-string 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 Python Standard Library.

Rest na gehele deling: gebruik van de module operator

Je kunt de modulo operator (`%`) gebruiken om de rest te bepalen na een gehele deling. Als beide operandi integers zijn, dan is het resultaat 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
```

Extra wiskundige functionaliteit: de `math` module

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

De `math` module is één van die modules uit de 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 de Python Standard Library.

Hoe controleert Dodona floating point getallen?

Als een opgave aangeeft dat er een *floating point* getal moet uitgeschreven worden, zonder expliciet het aantal cijfers na de komma te vermelden waarmee het getal moet uitgeschreven worden (zonder afronden of afkappen), dan zal Dodona 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.

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. Je kunt de vermenigvuldiging niet stilzwijgend uitvoeren.

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

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
```

Goniometrische functies uit de `math` module

De `math` module van de Python Standard Library definieert een aantal **goniometrische functie** zoals de sinusfunctie (`sin`), de cosinusfunctie (`cos`) en de tangensfunctie (`tan`). Belangrijk om weten is dat de hoeken die aan deze functies moeten doorgegeven worden in **radialen** moeten uitgedrukt worden en niet in graden. Gelukkig definieert de `math` module ook functies om een hoek in graden om te zetten naar radialen (`radians`) en omgekeerd (`degrees`).

```
>>> import math
>>> hoek = 90
>>> radialen = math.radians(hoek)
>>> radialen
1.5707963267948966
>>> radialen == math.pi / 2
True
>>> math.cos(radialen) # moet 0 opleveren, maar let op de afrondingsfout
6.123233995736766e-17
>>> math.sin(radialen)
1.0
```

Cyclometrische functies uit de `math` module

De `math` module van de Python Standard Library definieert een aantal **cyclometrische functies** (inverse functies van de 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
```

Geldigheid van bewerkingen hangt af van gegevenstypes

`TypeError: unsupported operand type(s) for ...`

Ga na of je er op gelet hebt dat de ingebouwde functie `input` altijd een string als resultaat teruggeeft. In veel gevallen is het nodig om dit resultaat om te zetten naar het gepaste gegevenstype dooréén van de ingebouwde functies voor typeconversies (`int`, `float`, `str`, ...) te gebruiken.

```
>>> leeftijd = input('Hoe oud ben je? ')
Hoe oud ben je? 3
>>> 10 + leeftijd
Traceback (most recent call last):
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> 10 + int(leeftijd)
13
```

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 af van het gegevenstype van de operandi. Als beide operandi integers zijn, dan is het resultaat ook een integer. Als één of beide operandi `float`s 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
```