Algemeen

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, dan 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.

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
```

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.

Notatie van floating point getallen

Floating point getallen worden in Python geschreven met een decimaal punt, niet met een komma. Komma's zijn in Python voorbehouden om argumenten van een functie of elementen van een samengesteld gegevenstype van elkaar te scheiden.

```
>>> 3.14159          # floating point getal
3.14159
>>> 3,14159          # tuple van twee integers
(3, 14159)
```

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
```

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
```

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
```

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 door gebruik te maken van één van de ingebouwde functies voor typeconversies (`int`, `float`, `str`, ...)

```
>>> 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
```

Wijzers van de klok

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 Python Standard Library.

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 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
```

Voorloopnullen toevoegen met f-strings

Als je bij het omzetten van een getal naar een string wil zorgen dat de string een vast aantal karakters heeft (waarbij er eventueel nullen worden toegevoegd aan het begin van de string), dan kan je gebruikmaken van een *format specifier*. *Format specifiers* worden altijd tussen het paar accolades geplaatst dat gebruikt wordt als plaatshouder in de template string. Een *format specifier* begint altijd met een dubbelpunt (`:`).

Om een getal uit te schrijven dat een vast aantal posities inneemt, gebruik je de *format specifier* `:0nd`. De `0` staat er omdat we vooraan nullen willen toevoegen (je kan ook andere karakters toevoegen), de letter `d` staat voor een digit (getal) en `n` voor de lengte die de string moet innemen. Als het getal dat je wil omzetten naar een string langer is dan de gewenste lengte, dan wordt het volledige getal overgenomen. In dat geval zal de string dus langer zijn dan het gewenst aantal karakters.

```
>>> f'{2}'
'2'
>>> f'{2:02d}'
'02'
>>> f'{34:02d]'
'34'
>>> f'{567:02d}'
'567'
>>> f'{89:06d}'
'000089'
```

Er zijn nog andere manieren om voorloopnullen te genereren. Zo kan je ook gebruikmaken van de stringmethode `zfill` (*zero fill*). Je kan ook het verschil tussen de huidige lengte en de gewenste lengte berekenen en dan weet je hoeveel nullen je moet toevoegen. Of je kan ook werken met een `while` lus die nullen blijft toevoegen totdat de string de gewenste lengte heeft.

```
>>> gewenste_lengte = 3
>>> getal = str(2)
>>> getal.zfill(gewenste_lengte)
'002'
>>> aantal_nullen = gewenste_lengte - len(getal)
>>> aantal_nullen
2
>>> '0' * aantal_nullen + getal
'002'
>>> while len(getal) < gewenste_lengte:
...     getal = '0' + getal
...
>>> getal
'002'
```

Kleinste waarde bepalen

De ingebouwde functie `min` kan gebruikt worden om het minimum van twee waarden te bepalen.

```
>>> min(7, 3)
3
>>> min(3.14, 7.45)
3.14
```

Dezelfde functie kan ook gebruikt worden om het minimum van meerdere waarden te bepalen.

```
>>> min(7, 3, 8, 19, 2, 12)
2
>>> min(3.14, 7.45, 17.35, 373.21, 2.34, 98.36)
2.34
```

Absolute waarde

De ingebouwde functie `abs` kan gebruikt worden om de absolute waarde van een getal te bepalen.

```
>>> abs(42)
42
>>> abs(-42)
42
>>> abs(3.14159)
3.14159
>>> abs(-3.14159)
3.14159
```

Specifieke info

Iedere positie van een wijzer op de klok kan uitgedrukt worden als het aantal graden van de hoek (in wijzerzin natuurlijk) die hij maakt met de richting die wordt aangegeven door 12 uur. Zo staat de grote wijzer om 3 uur bijvoorbeeld in een hoek van 90° ten opzichte van 12 uur.

De hoek tussen de twee wijzers kan dan berekend worden als het verschil van hun posities uitgedrukt in graden.

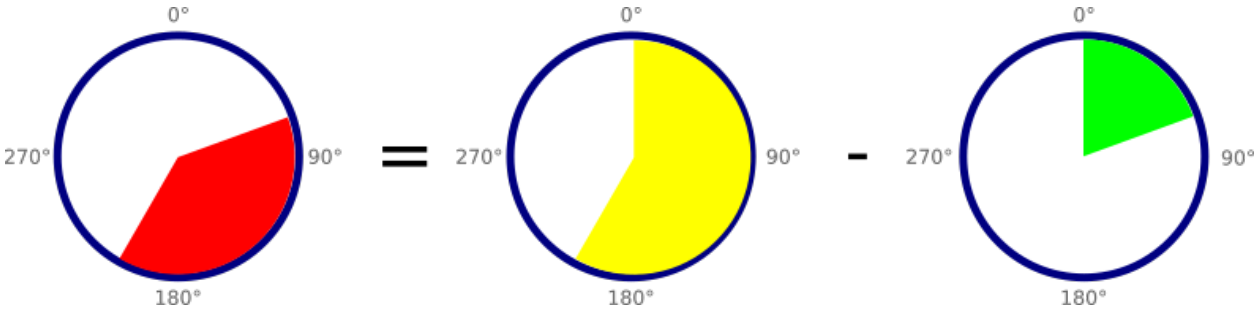

Figure 1: Hoeken