Algemeen

Opgemaakte tekst: de stringmethode `format`

Wanneer je een string op een bepaalde manier wil samenstellen uit vaste en variabele tekstfragmenten, dan kan het handig zijn om gebruik te maken van de stringmethode `format`. Je kunt deze methode aanroepen op een vaste string die als template fungeert. Elk variabel fragment wordt in de template aangeduid met een paar accolades (`{}`). Voor elk variabel fragment moet een argument doorgegeven worden aan de methode `format`. De methode zal de waarde van elk argument invullen op de corresponderende positie in de template, zodat een nieuwe (ingevulde) string ontstaat die door de methode wordt teruggegeven.

In onderstaand codefragment definiëren we bijvoorbeeld drie variabelen `x`, `y` en `som`, waarbij `som` berekend wordt als de som van de variabelen `x` en `y`. De code maakt gebruik van de stringmethode `format` om een string samen te stellen van de vorm `x + y  = som`, waarbij de posities `x`, `y` en `sum` worden ingevuld met de waarden van de gelijknamige variabelen. De string die wordt teruggegeven door de stringmethode `format` wordt dan uitgeschreven aan de hand van de ingebouwde functie print.

```
>>> x = 2
>>> y = 3
>>> som = x + y
>>> print('De som van {} en {} is {}'.format(x, y, som))
De som van 2 en 3 is 5
```

Een paar accolades in een templatestring wordt vaak een plaatshouder (of *placeholder* in het Engels) genoemd. Standaard wordt de eerste plaatshouder ingevuld met de waarde van het eerste argument dat wordt doorgegeven aan de stringmethode `format`, de tweede plaatshouder met de waarde van het tweede argument, enzoverder. We zeggen dan dat de plaatshouders *positioneel* ingevuld worden. De methode `format` biedt ondersteuning voor andere manieren om de plaatshouders in te vullen, en laat ook toe om meer gedetailleerde opmaak vast te leggen voor de waarden die op de posities van de plaatshouders moeten ingevuld worden. Voor meer details over de stringmethode `format` verwijzen we naar de The 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 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 de stringmethode `format`. Binnen een paar accolades dat gebruikt wordt als plaatshouder in de templatestring kan je immers opgeven hoe de ingevulde waarde precies moet opgemaakt worden. Dit doe je door 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.

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

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

Rest na gehele deling: gebruik van de module operator

In Python kan je gebruik maken 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
```

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 er 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.

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
```

Hartslagen

Algemene info

Opmerkingen

Machtsverheffing in Python

Net zoals Python operatoren heeft voor de optelling (`+`) en de vermenigvulding (`*`), is er ook een operator voor de machtsverheffing: `**`.

```
>>> 10 ** 2   # 10 tot de tweede macht
100
>>> 2 ** 3    # 2 tot de derde macht
8
```

De diatomist

Specifieke info

Een cirkel met straal r heeft omtrek $2\pi r$ en oppervlakte πr^2.

Tijdmeting op Mars

Algemene info

Gebruik van modulo en gehele deling voor omzetten van eenheden

Stel dat een integervariabele `seconden` een tijdsduur uitdrukt in seconden. Dan kan je deze tijdsduur op de volgende manier uitdrukken in uren, minuten en seconden.

```
>>> seconden = 123456

>>> uren = seconden // 3600
>>> minuten = (seconden // 60) % 60
>>> seconden = seconden % 60

>>> print('{} uren, {} minuten, {} seconden'.format(uren, minuten, seconden))
34 uren, 17 minuten, 36 seconden
```

Wijzers van de klok

Algemene info

Voorloopnullen toevoegen met stringmethode `format`

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 kun je gebruik maken van

de stringmethode `format` en een bijhorende * 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 lang is, gebruik je de specifier `:0nd`. De `0` staat er omdat we vooraan nullen willen toevoegen (je kan ook andere karakters toevoegen), de letter `d` staat voor digit (getal) en `n` voor de lengte die de string moet hebben. 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.

```
>>> "{}".format(2)
'2'
>>> "{:02}".format(2)
'02d'
>>> "{:02d]".format(34)
'34'
>>> "{:02d}".format(567)
'567'
>>> "{:06d}".format(89)
'000089'
```

Er zijn nog andere manieren om voorloopnullen te genereren. Zo kan je ook gebruik maken van de stringmethode `zfill`. 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'
```

Opmerkingen

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
```

Diezelfde 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.

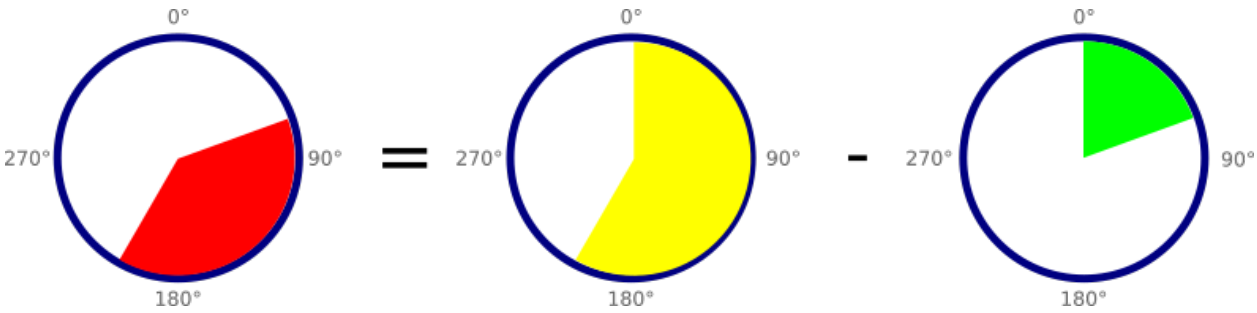

Figure 1: Hoeken