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.

Waar is de vader?

Algemene info

Opmerkingen

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
```

Mogelijke fouten

`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 typeconversiefuncties (`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
```

Specifieke info

Dit probleem kunnen we herleiden naar een stelsel met twee vergelijkingen en twee onbekenden. Stel dat we de leeftijd van de moeder (in maanden) aanduiden met de variabele m en dat we de leeftijd van haar zoon (in maanden) aanduiden met de variabele z. Dan kunnen we de volgende twee vergelijkingen opstellen:

- Een moeder is a jaar ouder dan haar zoon:

$$m = z + 12a$$

- Over b jaar zal de moeder c keer zijn leeftijd hebben:

$$m + 12b = c(z + 12b)$$

Let hierbij op het feit dat we de waarden a en b onmiddellijk vermenigvuldigen met 12, omdat we alles uitdrukken in maanden en de waarden a en b worden initieel opgegeven in jaren. We moeten dus het volgende stelsel vergelijkingen oplossen naar m en z

$$\begin{cases} m = z + 12a \\ m + 12b = c(z + 12b) \end{cases}$$

We kunnen bijvoorbeeld m uit de eerste vergelijking substitueren in de tweede vergelijking, zodat we de volgende uitdrukking bekomen

$$z + 12a + 12b = c(z + 12b)$$

waarin m volledig is weggewerkt. Als we deze vergelijking oplossen naar z dan bekomen we

$$\begin{aligned} cz + 12bc &= z + 12a + 12b \\ cz - z &= 12a + 12b - 12bc \\ z(c-1) &= 12(a + b - bc) \\ z &= \frac{12(a + b - bc)}{c - 1} \end{aligned}$$

Nu we de leeftijd van de zoon bepaald hebben, kunnen we die subsitueren in de eerste vergelijking om daarmee ook de leeftijd van de moeder te bepalen

$$m = z + 12a$$

Opmerking: Als extra tip geven we nog mee dat het afgeraden is om formules te proberen kopiëren uit een PDF bestand. Bij het kopiëren zullen immers niet alle tekens overeenkomen met de operatoren die je in Python broncode moet gebruiken.

Stopwatch baby

Algemene info

`if-else`-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
'groot'

>>> x = 190
>>> if x > 185:
...     lengte = 'groot'
... else:
...     lengte = 'klein'
...
>>> lengte
'groot'
```

Je kan hetzelfde realiseren door gebruik te maken van een `if-else`-expressie (ook een *voorwaardelijke expressie* genoemd). In tegenstelling tot een voorwaardelijke opdracht is dit dus een expressie (die een waarde oplevert) en geen statement. Hierdoor kan je een `if-else`-expressie zonder problemen gebruiken als rechterlid van een toekenningsopdracht. De bovenstaande interactieve sessie met hetzelfde resultaat korter kan schrijven als

```
>>> x = 100
>>> lengte = 'groot' if x > 185 else 'klein'
>>> lengte
'klein'

>>> x = 190
>>> lengte = 'groot' if x > 185 else 'klein'
```

```
>>> lengte
'groot'
```

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 uitgeprint 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('{} dan 5'.format(resultaat))
'kleiner dan 5'
```

Opmerkingen

Toekenningsopdracht vs. gelijkheidstest

In Python maak je bij een toekenningsopdracht gebruik van één enkel gelijkteken en bij een gelijkheidstest (nagaan of twee objecten dezelfde waarde hebben) van twee opeenvolgende gelijktekens. Om te testen of de variabele `x` gelijk is aan twee schrijf je dus

```
>>> if x == 2:    # correct
...     pass
```

en niet

```
>>> if x = 2:     # verkeerd
  File "<myscript.py>", line 1
    if x = 2:
         ^
SyntaxError: invalid syntax
```

Mogelijke fouten

In Python is het een fout om de waarde van een variabele op te vragen als die variabele nog niet gedefinieer is (er is nog geen waarde toegekend aan die variabele). Dit doet zich bijvoorbeeld voor in het onderstaande codefragment

```
>>> x = 4
>>> if x < 3:
...     var = 'ok'
...
>>> print(var)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'var' is not defined
```

waar bij het `print` statement de foutmelding gemaakt wordt dat de variabele `var` niet kan teruggevonden worden in de namespace. Als je de voorgaande code goed bekijkt, dan zie je dat de enkel een waarde wordt toegekend aan de variabele `var` indien de waarde van `x` groter is dan 3 (en dat is hier duidelijk niet het geval).

Vierkant

Specifieke info

De euclidische afstand tussen twee punten $a = (x_1, y_1)$ en $b = (x_2, y_2)$ is gedefiniëerd als

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Het is handig om te weten dat als twee afstanden gelijk zijn aan elkaar, hun kwadraten dat ook zijn. Het is dus voldoende om de waarde

$$(x_1 - x_2)^2 + (y_1 - y_2)^2$$

te berekenen.

Algemeen

Oneindige lussen

Let op voor oneindige lussen. Een oneindige lus is een lus die eindeloos blijft uitgevoerd worden: in de meeste gevallen gaat het om een `while`-lus waarbij de statements binnen de lus er nooit voor zorgen dat de voorwaarde uiteindelijk `False` wordt. Bekijk bij wijze van bijvoorbeeld eens het volgend stuk code

```
>>> i = 0
>>> a = 0
>>> while i < 4:
...     a += 1
```

Omdat het statement `a += 1` er nooit zal voor zorgen dat de initiële waarde van de variabele `i` groter dan of gelijk aan 4 wordt, zal de voorwaarde `i < 4` altijd de waarde `True` opleveren.

Tip: Als je werkt met Eclipse, dan kan je nagaan dat je programma aan het uitvoeren is, door het rode vierkantje bovenaan de Console. Als je op dit vierkantje klikt, dan forceer je om de uitvoering van het programma te stoppen.

Ronde getallen

Algemene info

Opmerkingen

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
```

Liftparadox

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 string-methode `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'
```

De conditionele ternaire operator

De meeste programmeertalen hebben een ternaire operator die de conditionele operator genoemd wordt. Het resultaat van een expressie die gebruik maakt van deze operator hangt af van een conditie `test`. In python is de syntax van deze operator de volgende:

```
result = <expr_true> if <test> else <expr_false>
```

Als de conditie `test` als True evalueert zal de expressie `expr_true` uitgevoerd worden. Als `test` als False evalueert zal de expressie `expr_false` uitgevord worden. Het resultaat van de uitgevoerde expressie zal terecht komen in de variabele `result`. Let er hierbij op dat de volgorde van de expressies verschilt van andere programmeertalen waar eerst de expressie `test` geschreven wordt.

Boeketje rozen

Algemene info

Lussen vroegtijdig afbreken

In Python kan je gebruik maken van de statements `break` en `continue` om een lus vroegtijdig af te breken. Deze statements worden echter algemeen aanzien als slechte programmeerstijl.

Een situatie waarin je een lus vroegtijdig zou willen afbreken, doet zich bijvoorbeeld voor als je op zoek moet gaan naar een oplossing door een aantal mogelijke gevallen uit te proberen, en te stoppen van zodra je één oplossing gevonden hebt. In plaats van te werken met `break` en `continue` kan je in dit geval beter werken met een variabele die aangeeft of de oplossing reeds gevonden werd

```
>>> gevonden = False
>>> while not gevonden:
...     if (oplossing gevonden): # "oplossing gevonden" stelt voorwaarde voor
...         gevonden = True
...
```

Van zodra de oplossing gevonden werd (hier aangegeven door het feit dat de voorwaarde *oplossing gevonden* de waarde `True` aanneemt), wordt de variabele `gevonden` op `True` gezet, waardoor uit de `while`-lus zal gesprongen wanneer de `while`-voorwaarde opnieuw geëvalueerd wordt na de huidige iteratiestap.