Algemeen

Verzamelingen en dictionaries in doctests

De elementen van een verzameling en de sleutels van een dictionary hebben geen vaste volgorde. Dit betekent dat twee verzamelingen of twee dictionaries gelijk zijn, ongeacht de volgorde waarin de elementen/sleutels voorkomen bij de definitie ervan.

```
>>> {1, 3, 2, 4} == {4, 3, 1, 2}
True
>>> {'A': 1, 'B': 2, 'C': 3} == {'B': 2, 'A': 1, 'C': 3}
True
```

Verzamelingen en dictionaries kunnen echter problemen opleveren als je werkt met doctests, omdat die bij het vergelijken van de verwachte en gegenereerde uitvoer als volgt te werk gaan: de verwachte uitvoer wordt uit de docstring geëxtraheerd als een string, en de waarde die door de functie wordt teruggegeven of die als resultaat bekomen wordt bij de evaluatie van een expressie, wordt omgezet naar een string. Deze twee strings worden met elkaar vergeleken (als string, niet als verzameling of als dictionary). Bij het vergelijken van strings speelt de volgorde van de karakters wel een rol, en hier zit net het probleem.

```
>>> d1 = {'A': 1, 'B': 2, 'C': 3}
>>> s1 = str(d1)   # uitgevoerd op computer 1
>>> d1
"{'A': 1, 'C': 3, 'B': 2}"
>>> d2 = {'A': 1, 'B': 2, 'C': 3}
>>> s2 = str(d2)   # uitgevoerd op computer 2
"{'A': 1, 'B': 2, 'C': 3}"
>>> d1 == d2
True
>>> s1 == s2
False
```

Hoewel de dictionaries `d1` en `d2` dezelfde waarde hebben, geeft de doctest aan dat de stringvoorstellingen van deze twee dictionaries verschillend zijn. Het omzetten van een dictionary naar een string kan immers afhangen van de computer waarop je de code uitvoert, van de versie van Python die gebruikt wordt en kan zelfs verschillen als je het een paar keer na elkaar uitvoert op dezelfde computer met dezelfde Python versie. Het probleem kan opgelost worden door ervoor te zorgen dat de doctest geen strings maar elkaar vergelijkt, maar rechtstreeks de verzamelingen of dictionaries met elkaar vergelijkt. Als je dus oorspronkelijk een doctest krijgt van de volgende vorm

```
>>> functie(parameter1, parameter2)
{'A' : 1, 'B': 2, 'C': 3}
```

dan kan je deze doctest beter herschrijven als

```
>>> functie(parameter1, parameter2) == {'A' : 1, 'B': 2, 'C': 3}
True
```

Indien je je oplossing indient op Dodona, dan zullen de resultaten wel degelijk als verzamelingen of dictionaries geïnterpreteerd worden, en niet als strings. Daar doet het probleem zich dus niet voor.

Geslachtsverandering

Algemene info

Opmerkingen

Stringmethode `capitalize()`

De stringmethode `capitalize` kan gebruikt worden om het eerste karakter van een string om te zetten naar een hoofdletter (als het een letter is), en alle overige karakters naar kleine letters (als het letters zijn).

```
>>> 'aBcD'.capitalize()
'Abcd'
```

De heilige birmaan

Algemene info

Waarden toekennen aan sleutels in een dictionary

Bij het opbouwen van een dictionary is het soms nodig om een waarde aan een bepaalde sleutel toe te kennen, rekening houdend met feit dat er eventueel reeds een waarde met die sleutel geassocieerd is. Hierbij is het vaak zo dat je een nieuw sleutel/waarde paar moet toevoegen indien de sleutel nog niet gebruikt werd in de dictionary, en dat je een bestaand sleutel/waarde paar moet bijwerken indien de sleutel wel reeds gebruikt werd in de dictionary.

Figure 1: updaten dictionary

Deze techniek kan onder meer gebruikt worden om frequentietabellen op te bouwen. Frequentietabellen zijn dictionaries waarvoor iedere sleutel een waarde heeft die overeenkomt met het aantal keer dat die sleutel voorkomt in een containerobject (bv. een lijst, een tuple of een verzameling).

```
>>> lijst = ['R', 'S', 'E', 'E', 'N', 'T', 'E', 'I', 'L', 'D', 'I']
>>> frequentietabel(lijst)
{'E': 3, 'S': 1, 'D': 1, 'N': 1, 'T': 1, 'R': 1, 'L': 1, 'I': 2}
```

Bovenstaande techniek kan gebruikt worden om de functie `frequentietabel` te implementeren.

```
def frequentietabel(lijst):
    tabel = {}                  # lege dictionary aanmaken
    for element in lijst:
        if element not in tabel:
            tabel[element] = 0  # element toevoegen aan dictionary met initiële waarde 0
        tabel[element] += 1     # waarde geassocieerd met element bijwerken
```

In dit geval kan je echter ook gebruik maken van de dictionary-methode `get`. Deze methode vraagt de waarde op die in de dictionary geassocieerd is met een bepaalde sleutel. In tegenstelling tot het indexeren van een dictionary aan de hand van vierkante haakjes om de waarde op te halen die geassocieerd is met een bepaalde sleutel, zal de methode `get` niet resulteren in een `KeyError` indien de sleutel niet voorkomt in de dictionary. In plaats daarvan zal de methode `get` standaard de waarde `None` teruggeven. Als je een waarde doorgeeft aan de tweede optionele parameter, dan zal die waarde als standaardwaarde teruggegeven worden als de methode `get` de sleutel niet terugvindt in de dictionary.

```
>>> d = {'A': 1, 'B': 2, 'C': 3}
>>> d['A']
1
>>> d.get('A')
1
>>> d['D']
Traceback (most recent call last):
KeyError: 'D'
>>> d.get('D')                  # geeft de waarde None terug
>>> d.get('D', 0)
0
```

Pak me dan als je kan

Algemene info

Variabel aantal benoemde argumenten

Soms wil je dat de argumenten die een functie meekrijgt niet op voorhand vastliggen. Het gebruik van de dubbele ster als operator voor de naam van een parameter van een functie laat dit toe.

```
>>> def functie(**kwargs):
...     for sleutel in kwargs:
...         print("De parameter '{}' heeft waarde '{}'.".format(sleutel, kwargs[key]))
...
>>> functie(naam='Tim', leeftijd=8, woonplaats='Gent')
De parameter 'naam' heeft waarde 'Tim'.
De parameter 'leeftijd' heeft waarde '8'.
De parameter 'woonplaats' heeft waarde 'Gent'.
```

De variabele `kwargs` (zonder dubbele ster) kan geïnterpreteerd worden als een dictionary waaraan je, gegeven een naam van de parameter, de corresponderende waarde kunt opvragen.

Soms is het ook handig om in plaats van aan een functie een reeks benoemde parameters mee te geven, een dictionary mee te geven waarvan de sleutels overeenkomen met de namen van de parameters en de waarden met de corresponderende waarden van die parameters.

```
>>> def functie(naam, leeftijd, woonplaats):
...     print("{} is {} jaar oud en woont in {}.".format(naam, leeftijd, woonplaats))
>>> kwargs = {'naam': 'Tim', 'leeftijd': 8, 'woonplaats': 'Gent'}
>>> functie(**kwargs)
Tim is 8 jaar oud en woont in Gent.
```

Specifieke info

In de omschrijving van deze opgave staat dat voor een lineaire vergelijking met

$$n \in \mathbb{N}$$

parameters waarvan de waarden van

$$n - 1$$

parameters gekend zijn (met

$$n \geq 2$$

), de waarde van de ontbrekende parameter direct kan afgeleid worden uit de formule. Dit geldt in het bijzonder voor de jongleervergelijking. Om jullie het rekenwerk te besparen, vind je hieronder de jongleervergelijking opgelost naar elke parameter.

$$\begin{aligned} B &= \frac{H(F+D)}{V+D} \\ H &= \frac{B(V+D)}{F+D} \\ F &= \frac{B(V+D)}{H} - D \\ V &= \frac{H(F+D)}{B} - D \\ D &= \frac{BV-HF}{H-B} \end{aligned}$$