Algemeen

Dictionaries gebruiken om `if` statements te vermijden

Soms heb je in je code zeer lange `if-elif-else` statements waarbij alle gevallen eigenlijk hetzelfde doen, maar er één variabele is die verschilt voor elk geval. In dergelijke gevallen is het meestal beter om de waarde van die variabele op te zoeken in een dictionary en de geassocieerde waarde te gebruiken als resultaat.

```
>>> if x == 'A':
...     y += 3
... elif x == 'B':
...     y += 1
... elif x == 'C':
...     y += 7
```

Deze code kan korter geschreven worden door gebruik te maken van een dictionary (`dict`). Hierbij zijn de sleutels van de dictionary de verschillende waarden voor `x` en de waarde die bij iedere sleutel hoort, komt overeen met de waarde die bij `y` opgeteld moet worden.

```
>>> verhoging = {'A': 3, 'B': 1, 'C': 7}
>>> y += verhoging[x]
```

Variabel aantal benoemde argumenten

Soms wil je dat de argumenten die een functie meekrijgt vooraf niet 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(f"De parameter '{sleutel}' heeft waarde '{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) is een dictionary waaraan je de waarde van een gegeven parameter kan 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(f'{naam} is {leeftijd} jaar oud en woont in {woonplaats}.')
>>> kwargs = {'naam': 'Tim', 'leeftijd': 8, 'woonplaats': 'Gent'}
>>> functie(**kwargs)
Tim is 8 jaar oud en woont in Gent.
```

Een element uit een verzameling halen

Om een niet nader bepaald element uit een verzameling te halen, kan gebruikgemaakt worden van de methode `str.pop()`. Hierbij zal er een element uit de verzameling verwijderd en teruggegeven worden. Merk hierbij op dat je vooraf niet weet welk element dit zal zijn, maar dat dit ook niet echt willekeurig is.

```
>>> verzameling = {'ABC', 'abc', '123'}
>>> verzameling.pop()
'abc'
```

```
>>> verzameling
{'123', 'ABC'}
```

Wanneer je een bepaald element uit een verzameling wil halen, dan kan je daarvoor de methode `set.remove()` gebruiken.

```
>>> verzameling = {'ABC', 'abc', '123'}
>>> verzameling.remove('123')
>>> verzameling
{'abc', 'ABC'}
```

ISBN

Waarden toekennen aan de sleutels van 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 al 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 al 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 collectie (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}
```

Deze 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 gebruikmaken van de methode `dict.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 `dict.get()` niet leiden tot een `KeyError` als de sleutel niet voorkomt in de dictionary.

In plaats daarvan zal de methode `dict.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 `dict.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
```

Busroddels

Operatoren voor verzamelingen

In Python is het eenvoudig om de unie, de doorsnede en het verschil te bepalen van twee verzamelingen.

```
>>> verzameling1 = {'A', 'B', 'C'}
>>> verzameling2 = {'C', 'D', 'E'}
>>> verzameling1 & verzameling2        # doorsnede
{'C'}
>>> verzameling1 | verzameling2        # unie
{'A', 'B', 'C', 'D', 'E'}
>>> verzameling1 - verzameling2        # verschil
{'A', 'B'}
>>> verzameling2 - verzameling1        # verschil is asymmetrisch
{'D', 'E'}
```

Zonnestelsel

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 met 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
```

Als 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.

Specifieke info

In Python is het toegelaten om een functie te definiëren binnen een andere functie. Hierdoor is die functie enkel zichtbaar binnen de functie waarin ze gedefinieerd is. Bovendien heeft deze functie ook toegang tot alle lokale variabelen van de functie waarin ze gedefinieerd wordt.

Hieronder definiëren we bijvoorbeeld de functie `g` binnen de functie `f`, waarbij de functie `g` ook de variabele `lijst` kan aanspreken die binnen de functie `f` gedefinieerd wordt.

```
>>> def f():
...
...     lijst = ['a']
...
...     def g():
...         print(lijst)  # variable uit scope van f aanspreken
...
...     g()
...     lijst.append('b')
...     g()
...
>>> f()
['a']
['a', 'b']
```

In deze opgave kan je bijvoorbeeld van deze techniek gebruik maken om een functie te schrijven die je doorgeeft aan de parameter `key` van een sorteerfunctie (`sort` of `sorted`). Aan deze functie wordt immers maar één argument doorgegeven (een element van de lijst), terwijl je soms ook nog andere informatie nodig hebt om de sorteerwaarde van dat argument te bepalen.

```
>>> def sorteersleutel(waarde, volgorde):
...     return volgorde.index(waarde)
...
>>> def rangschikken(lijst, volgorde):
...     return sorted(lijst, key=sorteersleutel)
...
>>> rangschikken(['a', 'b', 'c'], 'cbad')
TypeError: sorteersleutel() missing 1 required positional argument: 'volgorde'
```

Het probleem is hier dat de ingebouwde functie `sorted` de functie `sorteersleutel` die wordt doorgegeven aan de parameter `key` aanroept met één enkel argument (een element van de lijst), terwijl de functie `sorteersleutel` met twee argumenten moet aangeroepen worden. Een mogelijke oplossing bestaat erin om de functie `sorteersleutel` te definiëren binnen de functie `rangschikken`.

```
>>> def rangschikken(lijst, volgorde):
...
...     def sorteersleutel(waarde):
...         return volgorde.index(waarde) # gebruik variabele volgorde uit scope van functie rangschikk
...
...     return sorted(lijst, key=sorteersleutel)
...
>>> rangschikken(['a', 'b', 'c'], 'cbad')
['c', 'b', 'a']
```

Omdat de variabele `volgorde` een lokale variabele is van de functie `rangschikken` (parameters zijn ook lokale variabelen), hoef je de waarde van die variabele dan niet noodzakelijk nog eens door te geven aan de functie `sorteersleutel`. Omdat de functie `sorteersleutel` binnen de functie `rangschikken` gedefinieerd wordt, heeft die ook rechtstreeks toegang tot de waarde van de variabele `volgorde`.