Algemeen

Hergebruik van bestaande functionaliteit

Functies zijn controlestructuren die handig zijn bij het vermijden van **codeduplicatie**. Code duplicatie is het fenomeen waarbij hetzelfde of een gelijkaardig stuk code op twee of meer plaatsen in je broncode voorkomt. Het is altijd een goed idee om codeduplicatie te vermijden.

Hou hierbij ook rekening met het feit dat je vanuit een functie ook andere functies kunt aanroepen. Soms staat het expliciet vermeld in de opgave dat je bij de implementatie van een functie, een andere functie (die je eerder reeds geïmplementeerd hebt) moet hergebruiken. Maar ook als een dergelijke expliciete vermelding niet in de opgave staat, blijft het de bedoeling dat je op zoek gaat om zoveel mogelijk bestaande functionaliteit (die je bijvoorbeeld in andere functies hebt geïmplementeerd) probeert te hergebruiken bij het implementeren van je functies.

Stel bijvoorbeeld dat je gevraagd wordt om twee functies te implementeren: `maxsom` en `minverschil`. Aan de eerste functie `maxsom` moeten drie argumenten doorgegeven worden, en moet de functie een Booleaanse waarde teruggeven die uitdrukt of de som van de eerste twee argumenten al dan niet kleiner is dan de waarde van het derde argument. Aan de tweede functie `minverschil` moeten drie argumenten doorgegeven worden, en moet de functie een Booleaanse waarde teruggeven die uitdrukt of de absolute waarde van het verschil van de eerste twee argumenten al dan niet groter is dan de waarde van het derde argument. Deze twee functies kunnen op de volgende manier geïmplementeerd worden.

```
def maxsom(x, y, a):
    return x + y < a

def minverschil(x, y, b):
    return abs(x - y) > b
```

Als je nu bovendien ook nog een derde functie `minmax` moet schrijven waaraan vier argumenten moeten doorgegeven worden, en die moet nagaan of de som van de eerste twee argumenten kleiner is dan het derde argumenten, en de absolute waarde van het verschil van de eerste twee argumenten groter is dan de waarde van het vierde argument, dan zou je deze functie op de volgende manier kunnen implementeren.

```
def minmax(x, y, a, b):
    return x + y < a and abs(x - y) > b
```

Hiermee ben je natuurlijk het *warm water* opnieuw aan het uitvinden. In dit geval is het impliciet de bedoeling dat je de functie `minmax` kunt implementeren door hergebruik te maken van de implementaties van de functies `maxsom` en `minverschil`. Op die manier kan je de implementatie van de functie eenvoudigweg als volgt neerschrijven.

```
def minmax(x, y, a, b):
    return maxsom(x, y, a) and minverschil(x, y, b)
```

Als je nu een wijziging zou aanbrengen aan je implementatie van de functie `maxsom` (omdat je bijvoorbeeld een efficiëntere oplossingsmethode gevonden hebt, of omdat er een *bug* in de implementatie bleek te zitten), dan moet je dit maar op één plaats aanpassen, en niet op twee plaatsen als je de code ook had gekopieerd in de functie `minmax`.

Functies/methoden: return vs print

Bij opgaven waarbij er gevraagd wordt om functies om methoden te implementeren, moet je heel goed opletten of er gevraagd wordt dat de functie/methode een resultaat moet **teruggeven**, dan wel dat de

functie/methode een resultaat moet **uitschrijven**. Om een functie/methode een resultaat te laten teruggeven maak je gebruik van van een `return` statement. Om een functie een functie/methode een resultaat te laten uitschrijven maak je gebruik van de ingebouwde functie `print`.

In de opgave C-som (reeks 05) wordt er bijvoorbeeld gevraagd om een functie `csom` te schrijven die een resultaat moet **teruggeven**. Een mogelijke correcte implementatie van deze functie is

```
def csom(getal):
    return getal % 9
```

waarbij een `return` statement gebruikt wordt om een berekende waarde te laten teruggeven door de functie. Stel dat we in plaats van een waarde terug te geven, verkeerdelijk gebruik zouden maken van een `print` statement om de berekende waarde uit te schrijven, en dat we de volgende oplossing zouden indien voor deze opgave.

```
def csom(getal):
    print(getal % 9)
```

Deze oplossing zal een **verkeerd antwoord** opleveren, waarbij we de volgende informatie te zien krijgen op de feedbackpagina.

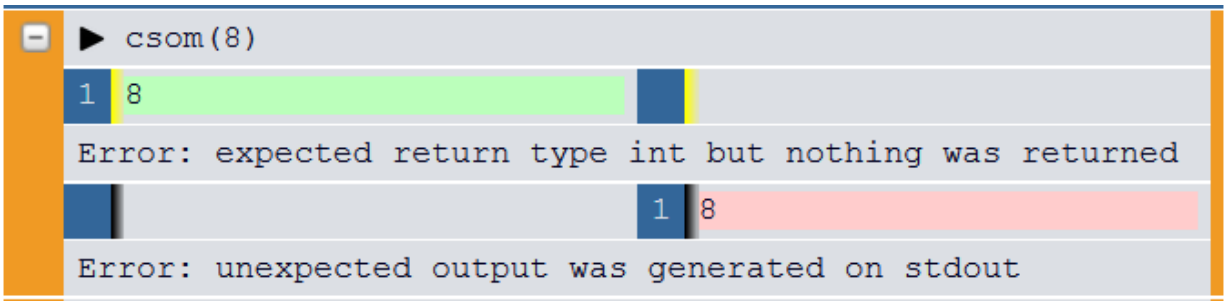

Figure 1: return vs print

Hierbij worden twee opmerkingen geformuleerd. De eerste opmerking geeft aan dat er verwacht werd dat de functie een integer waarde zou teruggeven (de waarde 8), maar dat de functie niet heeft teruggegeven (of meer specifiek, dat de functie de waarde `None` heeft teruggegeven). Dit is de betekenis van de foutmelding

```
Error: expected return type int but nothing was returned
```

Bovendien komt er nog een tweede opmerking die aangeeft dat de functie iets heeft uitgeschreven (naar *standard output*, of kortweg *stdout*) terwijl dat helemaal niet verwacht werd. Dit is de betekenis van de foutmelding

```
Error: unexpected output was generated on stdout
```

Als je een `return` statement had gebruikt in plaats van de `print` functie, dan waren beide foutmeldingen verdwenen en kreeg je een **correct antwoord**. Merk op dat resultaten die door een functie/methode worden teruggegeven in de feedbacktabel worden gemarkeerd met een gele band, en dat resultaten die door een functie/methode worden uitgeschreven worden gemarkeerd met een zwarte band.

Opkijken

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'
```