Algemeen

Controle of bepaalde voorwaarden gelden

Soms moet je in je programmacode expliciet nagaan of er aan bepaalde voorwaarden voldaan is, en moet je programma reageren als één van de voorwaarden geschonden is. Eén van de manieren waarop je dit kunt doen in Python is door gebruik te maken van het `assert` statement.

```
>>> x = 2
>>> y = 2
>>> assert x == y, 'beide getallen zijn niet gelijk'
>>> x = 1
>>> assert x == y, 'beide getallen zijn niet gelijk'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: beide getallen zijn niet gelijk
```

De algemene syntaxis van een `assert` statement is

`assert <voorwaarde>, <boodschap>`

Het `assert` statement controleert of er aan de voorwaarde voldaan is. Indien dat niet het geval is, dan zal er een `AssertionError` opgeworpen worden met de boodschap die wordt meegegeven op het einde van het `assert` statement. Indien deze uitzondering niet wordt opgevangen (wat voor deze cursus altijd het geval zal zijn), dan wordt op het punt van het opwerpen van de `AssertionError` de uitvoer van de code ook beëindigd (*runtime error*).

Kaprekarketen

De stringmethode `join`

De stringmethode `join` kan gebruikt worden om alle string in een *iterable object* (bv. een lijst) samen te voegen tot één enkele string. Hierbij worden alle strings van het *iterable object* samengevoegd, van elkaar gescheiden door een scheidingsteken waarop de stringmethode `join` wordt aangeroepen.

```
>>> lijst = ['a', 'b', 'c']
>>> ' '.join(lijst)
'a b c'
>>> ''.join(lijst)
'abc'
>>> '---'.join(lijst)
'a---b---c'
>>> ' - '.join(lijst)
'a - b - c'
```

Lijsten sorteren

In Python zijn er twee manieren om lijsten te sorteren. Ofwel roep je de lijstmethode `sort` aan op de lijst, ofwel geef je de lijst door aan de ingebouwde functie `sorted`. Er is echter wel een belangrijk verschil tussen deze twee opties. De lijstmethode `sort` wijzigt de lijst *in place* (en geeft geen nieuwe lijst terug), terwijl de ingebouwde functie `sorted` een nieuwe lijst teruggeeft waarvan de elementen van klein naar groot gesorteerd zijn.

```
>>> lijst = [4, 2, 3, 1]
>>> lijst.sort()
>>> lijst
[1, 2, 3, 4]
```

```
>>>
>>> lijst = [4, 2, 3, 1]
>>> sorted(lijst)
[1, 2, 3, 4]
```

Sorteren in dalende volgorde

Standaard sorteren de lijstmethode `sort` en de ingebouwde functie `sorted` de elementen van een lijst in stijgende volgorde. Beide hebben echter ook een optionele parameter `reverse` waaraan de waarde `True` kan doorgegeven worden om de elementen in dalende volgorde te laten sorteren.

```
>>> lijst = [1, 2, 3, 4]
>>> lijst.sort(reverse=True)
>>> lijst
[1, 2, 3, 4]
>>>
>>> lijst = [4, 2, 3, 1]
>>> sorted(lijst, reverse=True)
[1, 2, 3, 4]
```

Telefoonboekverdeling

Gegevenstype controleren met `isinstance`

Om na te gaan of een gegeven object een bepaald gegevenstype heeft, kan je natuurlijk gebruikmaken van de ingebouwde functie `type(o)` die het gegevenstype van het object `o` teruggeeft. Het is echter beter om hiervoor de ingebouwde functie `isinstance(o, t)` te gebruiken. Deze functie geeft een Booleaanse waarde terug die aangeeft of het object `o` al dan niet behoort tot het type `t`.

```
>>> type(3) == int
True
>>> isinstance(3.14, int)
False
>>> isinstance(3.14, float)
True
>>> isinstance([1, 2, 3], list)
True
```

Als je wil controleren of een gegeven object 1 van meerdere types is, kun je de volgende syntax gebruiken. Hierbij lijst je de mogelijke toegelaten types op in een tuple.

```
>>> isinstance(3, (str, int))
True
>>> isinstance('a', (str, int))
True
>> isinstance(['a'], (str, int))
False
```