General

Controle of bepaalde voorwaarden gelden

Sometimes it is needed to explicitly check if certain conditions hold when executing part of your source code, and the program needs to respond if one of the conditions is not met. One of the easiest ways this can be done in Python is by using the `assert` statement.

```
>>> x = 2
>>> y = 2
>>> assert x == y, 'the values are different'
>>> x = 1
>>> assert x == y, 'the values are different'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: the values are different
```

The general syntax of the `assert` statement is

```
assert <condition>, <message>
```

The `assert` statement checks whether or not the condition is met. If this is not the case, an `AssertionError` will be raised with the message that is given at the end of the `assert` statement. In case this exception is not caught elsewhere in the code (which will always be the case in this course), the execution of the codes halts at the point where the `AssertionError` was raised (*runtime error*).

Kaprekar series

The string method `join`

The string method `join` can be used to concatenate all string in an iterable object (e.g. a list) into a single string. This is done by concatenating all strings in the iterable object using a separator, which is the string on which the string method `join` is called.

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

Sorting lists

Python supports two ways to rearrange the elements of a list from the smallest to the largest. You can either call the list method `sort` on the list, or you can pass the list to the built-in function `sorted`. However, there is an important different between these two alternatives. The list method `sort` modifies the list *in place* (and does not return a new list), whereas the built-in function `sorted` returns a new list whose elements are sorted from the smallest to the largest.

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

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

Sort in descending order

By default the list method `sort` and the built-in function `sorted` sort the elements of a list in ascending order. Both function also have an optional parameter `reverse` to which the value `True` can be passed to have the elements arranged in descending order.

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

Partitioning the phone book

Check data types with `isinstance`

To check whether or not a given object `o` has a given data type `t`, you can use the built-in function `type(o)` to see if it returns the data type `t` for the object `o`. However, it is better (more pythonic) to use the built-in function `isinstance(o, t)` in this case. This function returns a Boolean value that indicates whether or not the object `o` has data type `t` or a date type that is derived from the data type `t`.

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

To check whether or not a given object `o` has one of multiple types, the following syntax can be used. All posible valid types are hereby listed in a tuple.

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