# Look and say

## Preparation

The iterator protocol of Python prescribes that iterable objects must support the following two methods:

iterator.__iter__()

> Returns the iterable object itself. This method is required to allow both containers and iterators to be used with the for and in statements.

iterator.__next__()

> Returns the next item from the iteration. If there are no further items, the StopIteration exception must be raised.

The following example shows how the class Reversed can be initialised with a sequence object (such as a string or a list). This class generates iterators that can be used to traverse the elements of the sequence object in reversed order.

```
class Reversed:

    """
    >>> for element in Reversed('python'):
    ...     print(element, end='')
    nohtyp
    >>> for element in Reversed([1, 2, 3, 4]):
    ...     print(element, end='')
    4321
    """

    def __init__(self, container):

        self.container = container
        self.index = len(container)

    def __iter__(self):

        return self

    def __next__(self):

        if self.index > 0:
            self.index -= 1
            return self.container[self.index]
        else:
            raise StopIteration
```

## Problem description

In mathematics, the look-and-say sequence is the sequence of integers beginning as follows:

> 1, 11, 21, 1211, 111221, 312211, 13112221, …

To generate a member of the sequence from the previous member, read off the digits of the previous member, counting the number of digits in groups of the same digit. For example:

- 1 is read off as "one 1" or 11
- 11 is read off as "two 1s" or 21
- 21 is read off as "one 2, then one 1" or 1211
- 1211 is read off as "one 1, then one 2, then two 1s" or 111221
- 111221 is read off as "three 1s, then two 2s, then one 1" or 312211

The sequence does not necessarily need to start with the number 1. The look-and-say sequence was introduced and analyzed by John Conway in his paper "The Weird and Wonderful Chemistry of Audioactive Decay" published in *Eureka* **46**, 5—18 in 1986.

## Assignment

Define a class `Conway` that can be used to initialize objects with a given strict positive integer. Objects of this class are iterators that follow the iterator protocol of Python. Each iteration should return the next number from Conway's look-and-say sequence, that follows the number that was returned previously. If no number was returned previously, the number must be returned that was used to initialize the object.

## Example

```
>>> sequence = Conway(1)
>>> next(sequence)
11
>>> next(sequence)
21
>>> next(sequence)
1211
>>> next(sequence)
111221
>>> next(sequence)
312211
>>> next(sequence)
13112221
>>> next(sequence)
1113213211

>>> sequence = Conway(333666999)
>>> next(sequence)
333639
>>> next(sequence)
33161319
>>> next(sequence)
23111611131119
```

## Voorbereiding

Het iteratorprotocol van Python schrijft voor dat iteratorobjecten de volgende twee methoden moeten ondersteunen:

```
iterator.__iter__()
```

Geeft het iteratorobject zelf terug. Deze methode is vereist, zodat zowel samengestelde gegevenstypes als iteratoren kunnen gebruikt worden binnen for en in statements.

iterator.__next__()

Geeft het volgende element terug uit de iteratie. Indien er geen elementen meer zijn, dan moet een StopIteration uitzondering opgeworpen worden.

Onderstaand voorbeeld illustreert hoe de klasse Omgekeerd kan geïnitialiseerd worden met een sequentie-object (bijvoorbeeld een string of een lijst). Deze klasse is een iterator die kan gebruikt worden om de elementen van het sequentie-object in omgekeerde volgorde te doorlopen.

```
class Omgekeerd:

    """
    >>> for element in Omgekeerd('python'):
    ...     print(element, end='')
    nohtyp
    >>> for element in Omgekeerd([1, 2, 3, 4]):
    ...     print(element, end='')
    4321
    """

    def __init__(self, container):

        self.container = container
        self.index = len(container)

    def __iter__(self):

        return self

    def __next__(self):

        if self.index > 0:
            self.index -= 1
            return self.container[self.index]
        else:
            raise StopIteration
```

## Probleemomschrijving

De rij van Conway werd geïntroduceerd door en vernoemd naar de Britse wiskundige John Conway. Het is een rij van natuurlijke getallen die als volgt begint:

1, 11, 21, 1211, 111221, 312211, 13112221, ...

Het volgende element van de rij is telkens een "beschrijving" van het vorige element. Vandaar dat de rij van Conway in het Engels ook wel de *look-and-say sequence* genoemd wordt. We hebben dus dat

- de beschrijving van 1 gegeven wordt door "één 1": 11
- de beschrijving van 11 gegeven wordt door "twee 1-en": 21
- de beschrijving van 21 gegeven wordt door "één 2 en dan één 1": 1211
- de beschrijving van 1211 gegeven wordt door "één 1, dan één 2 en dan twee 1-en":

111221

Een rij van Conway kan ook met een ander getal beginnen dan 1.

## Opgave

Schrijf een klasse `Conway` waarvan de objecten geïnitialiseerd worden met een gegeven natuurlijk getal. Objecten van deze klasse zijn iteratoren die het iteratorprotocol van Python volgen. Bij elke iteratie wordt het natuurlijke getal uit de rij van Conway teruggegeven dat volgt op het voorgaande natuurlijke getal dat werd teruggegeven. Indien er voorheen nog geen natuurlijk getal werd teruggegeven, dan moet het getal uit de rij van Conway teruggegeven worden dat volgt op het natuurlijke getal dat werd opgegeven bij het aanmaken van het object.

## Voorbeeld

```
>>> rij = Conway(1)
>>> next(rij)
11
>>> next(rij)
21
>>> next(rij)
1211
>>> next(rij)
111221
>>> next(rij)
312211
>>> next(rij)
13112221
>>> next(rij)
1113213211

>>> rij = Conway(333666999)
>>> next(rij)
333639
>>> next(rij)
33161319
>>> next(rij)
23111611131119
```