Lineup

De lijstmethode `insert`

De lijstmethode `append` kan gebruikt worden om een nieuw element toe te voegen aan het einde van een lijst. De lijst methode `insert` laat toe om een element toe te voegen op een aangegeven positie in de lijst. Indien de positie die wordt doorgegeven aan de lijstmethode `insert` groter dan of gelijk is aan de lengte van de lijst, dan wordt het element achteraan de lijst toegevoegd.

```
>>> lijst = []
>>> lijst.insert(0, 'a')
>>> lijst
['a']
>>> lijst.insert(0, 'b')
>>> lijst
['b', 'a']
>>> lijst.insert(1, 'c')
>>> lijst
['b', 'c', 'a']
>>> lijst.insert(10, 'd')
>>> lijst
['b', 'c', 'a', 'd']
```

Herschikking

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*).

Doemdagklok

Strings opsplitsen in verschillende delen

Om een string op te splitsen in verschillende delen, is het soms handig om gebruik te maken van de stringmethode `split`. Aan deze methode moet je een argument meegeven, dat aangeeft welke reeks opeenvolgende karakters moet gebruikt worden om de string (waarop je de methode aanroept) in stukken te breken. Deze reeks opeenvolgende karakters wordt vaak *het scheidingsteken* genoemd.

```
>>> string = 'a-b-c-d'
>>> string.split('-')
['a', 'b', 'c', 'd']
```

Standaard wordt de string op alle plaatsen gesplitst waar de reeks opeenvolgende karakters voorkomt, en worden alle delen waarin de string werd opgesplitst in een lijst gestopt, die door de methode wordt teruggegeven. Als je geen argument meegeeft aan de methode, dan splitst die standaard de string op waar er een reeks opeenvolgende witruimtekarakters voorkomt (spaties, tabs, newlines, ...). De stringmethode heeft ook nog een optioneel argument dat je kan gebruiken om aan te geven in hoeveel stukken de string maximaal moet opgesplitst worden.

Omdat de stringmethode `split` een lijst teruggeeft, kan je ook makkelijk rechtstreeks over deze lijst itereren met een `for`-lus.

```
>>> string = 'a-b-c-d'
>>> for element in string.split('-'):
...     print(element)
...
a
b
c
d
```

Kolomtranspositie

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]
```

De stringmethode `index`

De stringmethode `index` kan gebruikt worden om de meest linkse positie van een karakter in een string te bepalen.

```
>>> string = 'mississippi'
>>> teken = 'i'
>>> string.index(teken)
1
```

Dit codefragment vindt dus het eerste voorkomen van de letter `i` op positie 1 in de string `mississippi`. Let hierbij op het feit dat Python de posities van de karakters in een string indexeert vanaf 0, en niet vanaf 1.

Het valt wel op te merken dat deze methode niet altijd zeer efficiënt werkt. Om de positie te vinden van een karakter dat voor het eerst opduikt op plaats p, moet de functie de eerste $p - 1$ posities van de string doorlopen. Voor een karakter dat niet voorkomt in de string moet de stringmethode `index` zelfs de volledige string aflopen vooraleer er kan vastgesteld worden dat het karakter er niet gezien werd.

Specifieke info

Om de alfabetische volgorde van de letters in het gereduceerde sleutelwoord te berekenen kun je eerst het gereduceerde sleutelwoord sorteren en vervolgens de index van iedere letter opzoeken.

```
>>> gereduceerd = 'BANRM'
>>> gesorteerd = sorted(gereduceerd)
>>> gesorteerd
'ABMNR'
>>> gesorteerd.index('B')
1
>>> gesorteerd.index('A')
0
>>> gesorteerd.index('N')
3
>>> gesorteerd.index('R')
4
>>> gesorteerd.index('M')
2
```

De volgorde van de kolommen is dus

```
[1, 0, 3, 4, 2]
```

Koninginnen, paarden en pionnen

Specifieke info

Om te bepalen welke plaatsen bedreigd worden door de koningin, moeten er 8 richtingen gecontroleerd worden. Hiervoor kun je eerst een lijst aanmaken met alle mogelijke richtingen.

```
richtingen = [(0, 1), (1, 0), (0, -1), (-1, 0), (1, 1), (1, -1), (-1, 1), (-1, -1)]
```

Waarbij voor `(r, k) r` overeenkomt met de verandering in rij (1 is omlaag, -1 is omhoog) en `k` overeenkomt met de verandering in kolom (1 is naar rechts en -1 is naar links). Zo komt `(0, 1)` overeen met de positie rechts en `(1, -1)` met de positie links onder.

Vervolgens kun je een lus schrijven die de richtingen overloopt en gegeven de verandering in rij en kolom, het stuk code uitvoert dat bepaalt welke posities door een koningin bedreigd worden in die richting.