# Lineup

### The list method `insert`

The list method `append` can be used to add an element to the end of a list. The list method `insert` allows to add an element at a given position in a list. in case the position that is passed to the list method `insert` is greater than or equal to the length of the list, the element is appended at the end of the list.

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

# Recoupling

### 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*).

# Doomsday clock

### Split strings into multiple parts

Sometimes you need to split a string into multiple parts. One way to do this job makes use of the string method `split`. This method takes an optional string argument, that indicates the sequence of characters (called the *separator*) that needs to be used to split the string on which the method is called.

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

By default, the method splits the string at all positions where the separator occurs in the string, and places each of the parts in a list that is returned by the method. In case no argument is passed to the method, the string is split at each occurrence of a sequence of whitespace characters (spaces, tabs, newlines, ...). The string method also has another optional argument that you can use to indicate the maximal number of parts in which the string needs to be split.

Because the string method `split` returns a list, you can directly iterate over the elements of the list using a `for`-loop.

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

# Columnar transposition

### 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]
```

### The string method `index`

The string method `index` may be used to determine the position of the leftmost occurrence of a character in a strings.

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

This code fragment finds the leftmost occurrence of the letter `i` at position 1 in the string `mississippi`. Note that Python indexes the positions of the characters in a string starting from 0, not starting from 1.

It should be noted that this methode might be inefficient in case the position of the character can be derived directly. In order to find the position of a character that first occurs at position $p$, the index method must traverse the first $p-1$ positions of the string. For a character that does not occur in the string, the string method index must even travers all characters in the string before it can decide that the character was not observed.

### Specific information

To obtain the alphabetic order of the letters in the reduced keyword you can first sort the reduced keyword and thereafter look up the index of each letter.

```
>>> 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
```

Therefore the order of the columns is

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

## Queens, knights and pawns

### Specific information

To determine which positions are under attack by a queen, 8 directions must be checked. To do this you can create a lists of all possible directions

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

Where for `(r, c)` `r` matches the change in rows (1 is up, -1 is down) and `c` matches the change in columns (1 is right, -1 is left). This means `(0, 1)`matches the position one to the right and `(1, -1)`matches the position bottom left side.

Next you can write a loop that iterates over the directions and given the change in rows and colums, executes the peace of code that determines wich positions are under attack by the queen in that direction.