General

Sets and dictionaries in doctests

The elements of a set and the keys of a dictionary do not have a particular order. This means that two sets or two dictionaries are equal, irrespective of the order in which the elements/keys have been added to the set/dictionary.

```
>>> {1, 3, 2, 4} == {4, 3, 1, 2}
True
>>> {'A': 1, 'B': 2, 'C': 3} == {'B': 2, 'A': 1, 'C': 3}
True
```

When working with doctests, however, sets and dictionaries might cause you trouble. The reason for this problem is that in comparing expected and generated results, doctests proceed as follows: the expected result is extracted from the doctest as a string, and the value returned by a function or resulting from the evaluation of an expression is converted to a string. These two strings are then compared with each other (as a string, not as a set or a dictionary). In comparing strings, the order of the characters is important, and that's what's causing the trouble.

```
>>> d1 = {'A': 1, 'B': 2, 'C': 3}
>>> s1 = str(d1)   # executed on computer 1
>>> d1
"{'A': 1, 'C': 3, 'B': 2}"
>>> d2 = {'A': 1, 'B': 2, 'C': 3}
>>> s2 = str(d2)   # executed o computer 2
"{'A': 1, 'B': 2, 'C': 3}"
>>> d1 == d2
True
>>> s1 == s2
False
```

Although dictionaries `d1` and `d2` have the same value, the doctest indicates that the string representations of these two dictionaries are different. The conversion of a set/dictionary to a string may depend on the computer that executes the code, the Python version used on that computer, and might even differ if you do repeat the same conversion on the same computer using the same Python version. The problem can be solved by making sure the doctests do not compare strings, but directly compare sets or dictionaries. For example, if a doctest initially looks like

```
>>> aFunction(parameter1, parameter2)
{'A' : 1, 'B': 2, 'C': 3}
```

you may better rewrite this doctest as

```
>>> aFunction(parameter1, parameter2) == {'A' : 1, 'B': 2, 'C': 3}
True
```

This problem never occurs on Dodona, since its way of testing the source code never converts return values or results of expression evaluations into strings, but directly compares the resulting objects.

Changing gender

General information

Remarks

String method `capitalize()`

The string method `capitalize` can be used to convert the first character of a string to uppercase (if it is a letter) and all other characters to lowercase (if they are letters).

```
>>> 'aBcD'.capitalize()
'Abcd'
```

Sacred Cat of Burma

General information

Assign values to keys in a dictionary

When associating a value with a dictionary key, you may have to take into account that the dictionary may or may not already associate a value to this key. It is often the case that you have to add a new key/value pair in case the key was not used in the dictionary, and thay you have to update an existing key/value pair in case the key was already used in the dictionary.

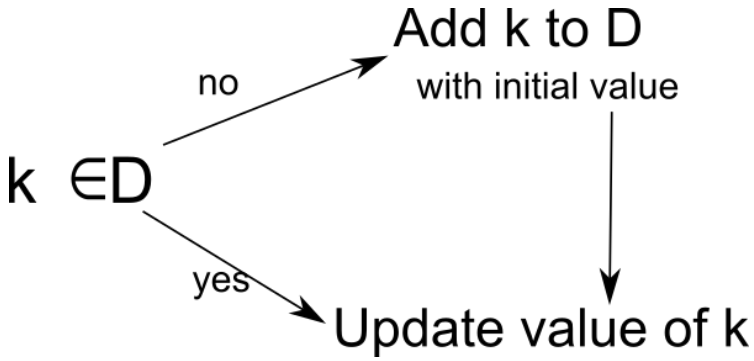

Figure 1: update dictionary

This technique can to be used, for example, for the construction of frequency tables. A frequency table is a dictionary that maps each key onto an integer that indicates how often the key occurs in a container object (e.g. a list, a tuple or a set).

```
>>> aList = ['R', 'S', 'E', 'E', 'N', 'T', 'E', 'I', 'L', 'D', 'I']
>>> frequentietabel(aList)
{'E': 3, 'S': 1, 'D': 1, 'N': 1, 'T': 1, 'R': 1, 'L': 1, 'I': 2}
```

The above technique can be used to implement the function `frequencyTable`.

```
def frequencyTable(aList):
    freq = {}                    # create empty dictionary
    for element in aList:
        if element not in freq:
            freq[element] = 0    # add element to dictionary with initial value 0
        freq[element] += 1       # update value associated with element
```

In this case you could have also used the dictionary method `get`. This method returns the value associated with a given key in the dictionary. In contrast to indexing dictionaries using square brackets to fetch the value associated with a given key, the `get` method will never result in a `KeyError` in case the key does not occur in the dictionary. Instead, the `get` method by default returns the value `None` if the key does not occur in the dictionary. If you pass a value to the second optional parameter, this value will be returned as the default value in case the `get` method does not find the key in the dictionary.

```
>>> d = {'A': 1, 'B': 2, 'C': 3}
>>> d['A']
1
>>> d.get('A')
1
>>> d['D']
Traceback (most recent call last):
KeyError: 'D'
>>> d.get('D')                   # returns the value None
>>> d.get('D', 0)
0
```

Catch as catch can

General information

Variable number of keyword arguments

Sometimes you won't like to fix the arguments of a function beforehand. The use of a double star as an operator in front of a function parameter allows you to accomplish this.

```
>>> def aFunction(**kwargs):
...     for key in kwargs:
...         print("The parameter '{}' has value '{}'.".format(key, kwargs[key]))
...
>>> aFunction(name='Tim', age=8, location='Ghent')
The parameter 'name' has value 'Tim'.
The parameter 'age' has value '8'.
The parameter 'location' has value 'Ghent'.
```

The variable `kwargs` (without the double star) now is a dictionary that maps parameter names onto their corresponding values.

Sometimes it's also handy not to pass a sequence of keyword arguments when calling a function, but instead wrap those arguments in a dictionary whose keys correspond to the parameter names.

```
>>> def aFunction(name, age, location):
...     print("{} is {} years old and lives in {}.".format(name, age, location))
>>> kwargs = {'name': 'Tim', 'age': 8, 'location': 'Ghent'}
>>> functie(**kwargs)
Tim is 8 years old and lives in Ghent.
```

Specific information

In the description of this assignment you'll find that if

$$n - 1$$

parameter values are known for a linear equation with

$$n \in \mathbb{N}$$

parameters (with

$$n \geq 2$$

), the missing parameter value can be computed directly from the formula. This especially holds for the juggling equation. To avoid that you'll have to do the derivations yourself, we have already solved the juggling equation for each of its parameters.

$$\begin{aligned} B &= \frac{H(F+D)}{V+D} \\ H &= \frac{B(V+D)}{F+D} \\ F &= \frac{B(V+D)}{H} - D \\ V &= \frac{H(F+D)}{B} - D \\ D &= \frac{BV-HF}{H-B} \end{aligned}$$