General

Reuse existing functionality

Functions are control structures that allow to avoid unnecessary **code duplication**. Code duplication is the phenomenon where multiple copies of the same or highly similar code occur in the source code of a program. It is always a good idea to avoid code duplication.

Also take into account the possibility to call other functions while implementing a function. Sometimes it will be explicitly stated in the description of an assignment that you have to reuse an existing function (one that you implemented earlier) in the implementation of a new function. But in other cases such an statement will not be made explicit in de description of the assignment, while it implicitly remains a goal to detect possible code reuse while implementing the functions.

Say for example that you were asked to implement two functions: `maxsum` and `mindiff`. The first function `maxsum` takes three arguments, and needs to return a Boolean value that expresses whether or not the sum of the first two arguments is less than the value of the third argument. The second function `mindiff` takes three arguments, and needs to return a Boolean value that expresses whether or not the absolute value of the difference of the first two arguments is larger than the value of the third argument. Both functions can be implemented as follows.

```
def maxsum(x, y, a):
    return x + y < a

def mindiff(x, y, b):
    return abs(x - y) > b
```

Now, say that you are also asked to implement a third function `minmax` that takes four arguments, and needs to return a Boolean$ value that expresses whether or not the sum of the first two arguments is less than the value of the third argument AND the absolute value of the difference of the first two arguments is larger than the value of the fourth argument. You could implement this function in the following way.

```
def minmax(x, y, a, b):
    return x + y < a and abs(x - y) > b
```

However, this implementation completely *reinvents the wheel* since the condition that needs to be check in this function is nothing but the composition of the two conditions that need to be checked in the functions `maxsum` and `mindiff`. As a result, it is a far better solution to implement the function `minmax` in the following way.

```
def minmax(x, y, a, b):
    return maxsum(x, y, a) and mindiff(x, y, b)
```

In case there is a need to make a modification to your implementation of the function `maxsom` (because you have found out there is a more efficient strategy for the implementation, or the initial implementation contained a *bug*), you only have to make the adjustments at a single location in your source code, and not in two locations if you had copied the source code for the implementation of the function `minmax`.

Functions/methods: return vs print

In assignments where you are asked to implement functions or methods, you should read carefully if the function either needs to **return** a result, or if the function needs to **print** a result. A `return` statement must be used to let the function return a result. The built-in function `print` must be used to let the function print a result to *standard output* (short: *stdout*).

In the assignment C-sum (series 05) you are asked, for example, to write a function `csum` that must **return** a result. One possible correct implementation of this function is

```
def csum(number):
    return number % 9
```

where a `return` statement is used to have the function return a computed value. Suppose that we erroneously used the built-in function `print` to write the computed value to *stdout*, and submitted the following incorrect solution for this assignment.

```
def csum(number):
    print(number % 9)
```

In this case, the Dodona platform would evaluate the submission as a **wrong answer**, where the following feedback would be given on the feedback page.

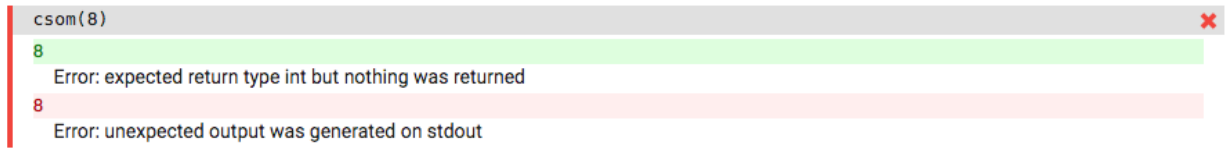

Figure 1: return vs print

The feedback contains two remarks. The first remark indicates that the function was expected to return an integer value (8), but instead the function did not return any value (or more precisely, the function returned the value `None`). This is the meaning of the error message

`Error: expected return type int but nothing was returned`

In addition, a second remark indicates that the function has written some information to *stdout*, whereas no information was expected on this output channel. This is the meaning of the error message

`Error: unexpected output was generated on stdout`

If you would have used a `return` statement instead of the built-in function `print`, both error message would have disappeared and the Dodona platform would have evaluated the submission as a **correct answer**. Also note that results returned by a function/method are marked with a thick yellow line in the feedback table, whereas results that are printed by a function/method are marked with a thick black line.

Transitions and transversions

General information

Working with *floating point* numbers in doctests

If a function returns a *floating point* number, this might give trouble when testing the correctness of the function using a doctest. This is caused by the fact that doctests perform an exact match between the string that represents the result in the doctest, and the result that is printed or returned by the function. In order to do this, the result of the function is first converted into a string. In comparing these two strings, doctests thus do not take into account the possibility of rounding errors that might occur when working with *floating point* numbers. These rounding errors are a consequence of the limited precision with which computers can represent real-valued numbers.

```
>>> 0.1
0.1
>>> 0.1 + 0.1 + 0.1
0.30000000000000004
```

If in executing a doctest the expected output (a string) does not exactly match the string representation that is returned by the function, the doctest will consider the result as incorrect.

The Dodona environment does take into account rounding errors when working with *floating point* numbers. Unless otherwise stated in the assignment, Dodona will check if the result is correct up to six decimal digits for functions that return *floating point* numbers (either directly or as elements of compound data types). This more or less comes down to rewriting a doctest according to the following strategy.

```
def multiply(x):

    """
    >>> abs(multiply(0.1) - 0.3) < 1e-6
    True
    """

    return 3 * x
```

The ouroboros dream

General information

Sorting the characters of a string

If you want to sort the letters of a string in alphabetic order, you can make use of the built-in function `sorted`. This function takes an iterable object (e.g. a string) and returns a list containing the elements of the iterable object in increasing order. For strings, this is a list containing the individual characters of the string, arranged according to the order of the characters in the ASCII table. As a result, letters are ordered in alphabetic order (but all uppercase letters come before all lowercase letters in the ASCII table).

```
>>> sorted('Python')
['P', 'h', 'n', 'o', 't', 'y']
>>> sorted('Python'.lower())
['h', 'n', 'o', 'p', 't', 'y']
```