Homework 1:
Python Basics, DocTests

Benjamin Roth, Marina Sedinkina
Symbolische Programmiersprache

Due: Thursday, October 24, 2019, 16:00

In this exercise you will:

- Review and implement some basic Python functionality.
- Get some hands-on experience using the python `doctest` framework for testing.

Exercise 1: Setting up the Git project

In order to have access to the Git project with the exercise code, and to be able to submit your solution, you need to do the following steps (ask the tutors if any of the steps is unclear to you):

1. Make sure you have a Gitlab account for gitlab2.cip.ifi.lmu.de
2. Form teams of 2 or 3 students (4 students are not allowed).
3. Use the web form (which you can find on sp1920.github.io) to provide the following information for your group:
   - Your team name
   - Gitlab id, name, and matriculation number of each team member
4. We will then create a project in Gitlab for you, that will contain the code to the exercises. Submit your solution by pushing to this project.
5. **Please do not create separate files or folders to submit your solution. Instead, change the files we provided.**
6. **Important: do NOT change the tests themselves, implement the missing functionality instead. Changing the tests will result in your exercise sheet scored with 0 points.**

Exercise 2: Python Basics

Exercise 2.1: Doctests

Use the doctest module to test your implementation of the functions in the module `hw01_basics.basics`.

Run your tests with (this assumes that you are in the src/ directory of your repository): `python3 -m doctest -v hw01_basics/basics.py`

Exercise 3: Python Basics

For each function, replace the pass statement, so that the function works properly as described below and indicated by the doctests. Make sure the doctests pass the test afterward.

1. `hello_semester()`

   Print the string '*Welcome to "Symbolische Programmierung" WS 19/20*'

2. `modulo(x,y):`

   Given two variables x and y. Calculate x mod y.

3. `odd(x):`

   Determine whether x is odd or not. Use the function `modulo(x,y)` in your calculation.

4. `happy_birthday(name, age):`

   Given a name and an age. Print the sentence "Happy >age<th birthday, >name<!".

5. `word_multiplier(word,n):`

   Given a word and a natural number. Return the word repeated itself n times in a row.

6. `reverse(w):`

   Return the reverse of a word, as you would read it backwards.

7. `every_nth(word, n):`

   Return only every *nth* letter of a word.

8. `second_element(list):`

   Return the second element from a list.

9. `concatenate_lists(list_a, list_b):`

   Return the concatenation of both lists.

10. `swap_half(list):`

    Swaps the first half of a list with the second half of the list. If the length of the list is odd, then the first half has one less element than the second half. You can make use of the *concatenate_lists(list_a, list_b)* function.

11. `replace_elements(list_a, replacement_indices, new_value):`

    Replace all elements in `list_a` at the positions in `replacement_indices` with the new value.

12. `long_strings(string_list, max_length):`

    Return a list of booleans: True for each string in `string_list` if the string is greater than `max_length` and False otherwise.

13. `print_squares(list):`

    Prints the square values of each element in the list. You can use a for-loop for this problem.

14. `count_to_k(k):`

    Print out the numbers counting from 0 to k, excluding k. If k is negative, count 'down' from 0, excluding 0. You can use a while-loop for this problem or the range(...) function.

15. `no_numbers(w):`

    Return True or False whether a string w contains no number symbols (0–9).

16. `contains_substring(string,substring):`

    Return True or False whether a string contains a substring.