Python (BSU FAMCS Fall'19)

Seminar 1

Advisor: Dzmitryi Kasitsyn

General notes

You can place your testing code under a conditional statement `if __name__ == '__main__'` at the end of the file. The code under the statement will be executed only if the file is a Python program entry point and will not if the file is imported.

I gently ask you to name your files and functions as it's specified in task.

Use Anytask system to submit your solutions. Just select proper task and attach your solutions (one or more .py files, not an archive) to start a reviewing process.

There are specific scripts to test solution interfaces. They're located nearby task statements.

Please remove all print statements that are used to debug your programs. You can use configurable logging facilites and unit tests instead.

Task 0. Setup and configure Python.

Setup a virtual environment. You can use *pip* (refer to `https://packaging.python.org/tutorials/installing-packages/#creating-virtual-environments`) or *PyCharm* IDE to do that.

Try out at least two different IDEs: PyCharm and Jupyter Notebook. The same – you can use either *pip* (refer to `https://jupyter.org/install.html`) or PyCharm (see interpreter settings) to setup Jupyter Notebook.

Task 1. (1 point). Let's call a ticket to be lucky if sum of digits at odd positions of its number equals to sum of ones at even positions.

Write down a function that takes a ticket number as an argument and returns the closest lucky ticket number (any if there are 2 of them). The closest ticket number means that the absolute difference between it and a given one is minimized subject to «lucky» property.

A given ticket number is supposed to be an admissible natural $2k$-digit number (it has even number of digits) without leading zeros.

Note. Save your program into a file *ticket.py* and give the name *get_nearest_lucky_ticket* to your solution function.

Example

```
assert get_nearest_lucky_ticket(111111) == 111111
assert get_nearest_lucky_ticket(123321) == 123321
assert get_nearest_lucky_ticket(123320) == 123321
assert get_nearest_lucky_ticket(333999) == 334004
```

Task 2. (1.5 points). Implement a function that *merge*s two sorted sequences into one sorted too with computational complexity $O(n)$. Your function should correctly process at least `list` and `tuple` pairs. The result type must be the same as type of input : merge of two lists produces a new list and merge of two tuples produces a tuple.

It's not permitted to use some built-in sorting and/or merging functions.

Also try to avoid random access to a sequence by indices – iterate over it consuming items one by one.

Name your solution function *merge* and save it into a file *merge.py*.

Example

```
assert merge([1, 2, 7], [3]) == [1, 2, 3, 7]
assert merge((3, 15), (7, 8)) == (3, 7, 8, 15)
```

Task 3. (1.5 points). You're asked to verify bank card numbers using Luhn algorithm (`https://en.wikipedia.org/wiki/Luhn_algorithm`). Implement two functions to do that.

The first one should take a bank card number type of `int` and return `true` if it's correct and `false` otherwise. Don't use any conversions to strings in the function. Name it *check_card_number*.

The second one should take a card number type of `str` and return the same as the integer type implementation. Don't convert the whole input to an integer, only single digit conversions are permitted. Give name *check_card_number_str* to your function.

Also provide a function *generate_card_number* that effectively generates random bank card numbers as integers or strings (up to your choice). Suppose that you're intended to generate either Visa (it has leading «4» digit) or Mastercard (it has leading «5») card numbers and card type is provided as a function parameter.

Note. Save your functions in *card_number.py* file.

Example

```
assert check_card_number(5082337440657928)  # valid Mastercard card number
assert not check_card_number_str('4601496706376197')  # invalid Visa card number
```