Python (BSU FAMCS Fall'19)

Семинар 1

Преподаватель: Дмитрий Косицин

Общие замечания ко всем заданиям

Свой тестирующий код можно размещать под условием `if __name__ == '__main__'` внизу файла. Такой код выполнится только если запустить этот файл, но не импортировать его.

Просьба использовать те имена для функций и файлов, которые указаны в замечаниях к заданиям.

Для отправки заданий выберите в anytask нужную задачу и там к сообщению прикрепите свое решение (один или несколько .py файлов, не архив).

Для тестирования интерфейса ваших заданий выложен специальный скрипт. Просьба также не оставлять debug print'ы в ваших программах, которые нужны исключительно для вывода отладочной информации.

Задание 0. Установите и настройте Python.

Создайте виртуальное окружение: с помощью pip есть простая инструкция (`https://packaging.python.org/tutorials/installing-packages/#creating-virtual-environments`), а в PyCharm его можно создать при создании проекта.

Попробуйте поработать как минимум в двух средах разработки: PyCharm и Jupyter Notebook. Последний легко установить через pip (см. `https://jupyter.org/install.html`) или в настройках интерпретатора в PyCharm.

Задание 1. (1 балл). Билетик назовем счастливым, если сумма цифр на нечетных позициях его номера равна сумме цифр на четных позициях.

Напишите функцию, принимающую номер билета и возвращающую номер ближайшего (по модулю разности между числами) счастливого билета (если их два – то любой из них). Номер переданного билета полагайте допустимым целым $2k$-значным (с четным количеством разрядов) числом без нулей в начале.

Замечание. Программу сохраните в файле *ticket.py*, функцию назовите *get_nearest_lucky_ticket*.

Пример

```
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
```

Задание 2. (1.5 балла). Реализуйте функцию, которая объединяет две отсортированные последовательности в одну отстортированную (merge, асимптотическая сложность алгоритма – $O(n)$). Программа должна корректно работать для списков (возвращать новый отсортированный список) и кортежей (возвращать кортеж). Использовать встроенные функции сортировки и/или слияния запрещается.

Полный балл за задачу будет выставлен только в случае отсутствия итерирования по сортируемой коллекции с помощью индексов. Иначе говоря, попробуйте избегать произвольной индексации и использовать только методы объектов.

Функцию назовите *merge*. Программу сохраните в файле *merge.py*.

Пример

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

Задание 3. (1.5 балла). В данной задаче предлагается проверять корректность номеров банковских карт с помощью алгоритма Луна (`https://en.wikipedia.org/wiki/Luhn_algorithm`). На вход функции передается один параметр – номер банковской карты, результат функции – `true` (если номер корректный) или `false` (в противном случае).

Для реализации данной проверки предлагается реализовать две функции: одна работает с номером как с целым числом (не приводя его к строке), а вторая – как со строкой (не приводя ее к числу).

Также реализуйте эффективную функцию генерации случайных строк или чисел (на выбор) – номеров карт Visa (начинается обычно с цифры 4) и Mastercard (начинается с цифры 5). Тип карты задайте параметром функции.

Замечание. Программу сохраните в файле *card_ number.py*, функции назовите *check_ card_ number*, *check_ card_ number_ str* и *generate_ card_ number* соответственно.

Пример

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