**Лабораторная работа №2**

**Общее описание**

В ходе выполнения лабораторной работы студенту необходимо реализовать сериализатор. Получившийся сериализатор должен корректно сериализовывать (сохранять / упаковывать) и десериализовать (восстанавливать / распаковывать) хранимую информацию. И разработать на основе сериализатора консольную утилиту.

Код вашей программы должен содержать фабричный метод create\_serializer(), который будет порождать различные типы сериализаторов: JSON, YAML, TOML. Должна быть возможность легко добавить новый сериализатор, не изменяя архитектуру приложения.

Каждый из сериализаторов должен реализовывать следующие методы:

* dump(obj, fp) — сериализует Python объект в файл
* dumps(obj) — сериализует Python объект в строку
* load(fp) — десериализует Python объект из файла
* loads(s) — десериализует Python объект из строки

Дополнительные аргументы в методы можете передавать какие хотите :)

Сериализация/десериализация :

* класса
* объекта с простыми полями
* объекта со сложными полями и функциями
* функции

Консольная утилита должна работать следующим образом:

Конвертация сериализованных объектов из одного поддерживаемого формата в другой. Путь к файлу (файлам) указывается относительным или абсолютным путем, отдельным параметром передается новый формат. При указании исходного формата конвертирование не должно выполняться.

В случае передачи параметром файла конфигурации, вся информация должна браться оттуда и все остальные параметры проигнорированы.

**Требования к программе**

**Разрешается использовать только стандартную библиотеку Python.**

**Пишем на версии Python 3.8+**

Режимы работы:

* как библиотека, для переиспользования основной логики или вспомогательных функций
* как консольная утилита

Задание конфигурации:

* Возможность передачи всех конфигурационных параметров по отдельности через аргументы командной строки
* Опциональный конфигурационный файл для конкретного запуска через аргументы командной строки

Внутреннее устройство:

* Работа с аргументами командной строки с помощью модуля *argparse*
* Структура программы должна быть разбита на модули
* Реализовать программу так, чтобы отдельный полезный функционал можно было бы переиспользовать, пользуясь программой как библиотекой

Защита от ошибок и тестирование:

* Основная функциональность должна быть покрыта юнит-тестами. Тесты запускают программу в различных режимах работы и проверяют результаты. Примеры фреймворков: pytest, nose, unittest и другие какие хотите.
* Coverage должен быть 90+ %
* Если ваша программа падает во время тестов или при сдаче преподавателю, то лабораторная не засчитывается.

Установка:

* С помощью setup.py.

**Критерии приема и сдачи лабораторной** (Если хотя бы 1 пункт не выполняется, то лабораторная не засчитывается):

* Хорошие знания в теории по темам лабораторной
* Понимание того, что написано у вас в коде
* Покрытие тестами
* Примечание: сериализация объекта за счет сохранения его исходного кода и десериализация через eval - не засчитывается как выполненная работа.

**Теория и практика для реализации и защиты лабораторной**

1. Темы рассмотренные на лекциях.
2. Обратите внимание на реализацию библиотеки cloudpickle. Там можно подсмотреть идеи по реализации лабораторной. https://github.com/cloudpipe/cloudpickle
3. Для тех кто выберет реализацию через кастомный Pickler <https://docs.python.org/3/library/pickle.html#pickle.Pickler>
4. <https://docs.python.org/3/library/inspect.html#module-inspect>
5. P.S. ссылки на модули приведены для примера того через что можно реализовывать лабораторную